PC 8300

PERSONAL COMPUTER

CONTENT

CHAPTER 1

SETTING UP THE COMPUTER

Unpack package box of the computer that you have just purchased from an authorized dealer, you will find the computer main unit together with a AC adaptor, a TV cable, a pair of cassette cable, and this manual book. Fig. 1.1 shows a photo of the computer.

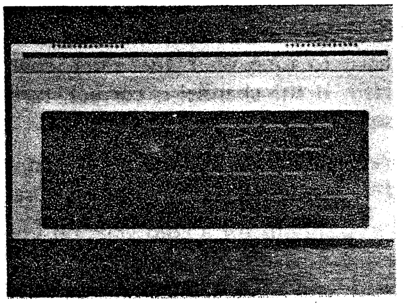


Fig. 1.1

HOOKING UP THE CRT DISPLAY

From the rear side of the computer (See Fig. 1.2), you can find a TV port and a monitor port. If a TV is being used as the computer's display, connect the TV port to the TV antenna coaxial input directly with the TV cable (Note: remove the external antenna from the TV port).

To obtain more stable and less distortion pictures, a monitor is recommended.

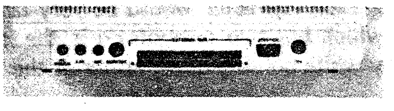


Fig. 1.2

Note: If your TV have a balancing antenna terminal (its resistance is 300 ohms) and not a coaxial antenna terminal (its resistance is 75 ohms), you must use an optional resistance converter to convert the resistance from 300 ohms to 75 ohms. Otherwise you cannot match the TV and the computer.

THE CASSETTE RECORDER

The cassette recorder cable connects the computer to any ordinary cassette recorder via the two cassette interface ports labeled "EAR" and "MIC" on the rear side. The recorder is used only when you are saving programs on tape or you are loading saved programs to your computer.

SETTING UP AC ADAPTOR

The last stage is to set up the AC adaptor. You can easily plug the adaptor into any AC power line, then plug the DC output of the adaptor into the computer DC input port (that you have seen from the rear side of the computer).

Inserting the DC plug will simply turn ON the computer. The power lamp at the upper right corner of the keyboard indicates whether the power is ON or OFF. You can also turn OFF the computer by unpluging the power cord from the computer jack. Be remember that any time you turn OFF the computer, the contents of the computer's memory will be erased. So be take care not to unplug the cord from the jack unless necessary. When you are not using the computer, it is best to unplug the adaptor from the AC power line. It is not a good idea to leave the live voltage around.

ADJUSTING THE SCREEN

Now turn down the TV volumn all the way, and turn ON your TV. Then you turn ON the computer by pluging the power cord into the computer DC input jack. If you have followed each step correctly, three things will happen when you turn ON the computer:

The power lamp should light. The computer's speaker should "beep". A blinking square called "cursor" should appear at the bottom left corner and letters "READY" at the upper left corner of the screen.

If the power lamp do light and the speaker do "beep" but you don't see the "READY" and blinking cursor on the screen, don't worry. You just have to tune your TV. First, check whether your TV is set to channel 3 UHF (or channel 2 UHF). If the channel is correct, turn the fine tuning knob of your TV set until you get a square cursor and letters "READY".

If you don't get the cursor and letters "READY", press the key marked RESET in the upper right corner of the keyboard. Your computer should "beep" when the RESET key is released.

If you still don't get the cursor, you should try a cold start on your computer. This can be done either by unpluging and pluging the power to your computer or by pressing the keys marked ENTER and RESET simultaneously and release the RESET first. You would properly get the cursor by doing this. (Note: Be remember that you erase all the contents of the memory by trying a cold start)

KEYBOARD

Study the keyboard. If you are familiar with standard typewriters, you will find a few differences between the keyboard and a typewriter keyboard. First, there are no lower case letters. You can get only capital letters on your computer. This is all you need for programming in your computer BASIC.

Using the diagram, locate the two SHIFT keys on the keyboard. The reason the keyboard has the SHIFT keys is to allow you for nearly twice as many characters with the same number of keys. A keyboard with a separate key for each character would be very large, making it hard to find any desired key. If you press a key which has two symbols on it, the lower symbol will appear on the screen. If you press the same key while holding down either of the SHIFT keys, the upper symbol will appear on the screen.

You may notice that on some of the keys, there are more than 2 symbols. For example on the key Q, there is a word SIN above the key and moreover there is a graphic symbol ▛ beside the letter Q. The letter Q is displayed if you press the key Q and the word SIN is displayed if you press the key Q while holding down the SHIFT key. To display the graphic symbol, you have to enter the graphic mode of the computer. To enter the graphic mode, just press the key with the word GRAPHIC on it. You have to press the SHIFT key at the same time because the word GRAPHIC is the upper symbol of the key. After pressing the key, one immediate response of the computer is that the cursor has changed. It is now a blinking letter G showing that you are in the graphic mode. Now press any key on your computer, you may notice that the computer displays the same characters, but in black on white (inverse video). To display the graphic symbols, press the key while holding the SHIFT down.

To exit from the graphic mode, just press the key ENTER. You see that the blinking letter G changes to the normal blinking cursor showing that the computer has back to normal mode.

When the Hindu mathematicians invented the open circle for the numeral zero, they didn't use the Roman alphabet. So they chose a symbol that, while not conflicting with their alphabet, looks just like our letter "O". The computer (and any straight-thinking individual) will want to keep zeros and oh's distinct. The usual method for doing this, on your computer and many other computers, is to put a slash through the zero. Now you can tell them apart. The keyboard and the TV display both make the distinction clear. Try them.

CHAPTER 2

BEGINNING BASIC

A FIRST LOOK AT THE PRINT STATEMENT

Now that you have the blinking cursor on the screen, you are ready to begin using the basic language. Type

    PRINT "HELLO"

and then press ENTER, the computer will print

    HELLO

at the top of the screen.

At the bottom left corner of the screen, some additional symbols appear. They make up a report message your computer displays after it processes the input. In this example the report message is OK which means that the computer successfully complete the task you ask it to do. The report message will remain until a key is pressed.

After a command is executed or after a program execution is completed or interrupted, a report message will be displayed showing what has happened and where in the program it happened. The table in the Appendix B gives all kinds of report message with a general description and a list of the situations where it can occur.

The statement

    PRINT "HELLO"

is an instruction to the computer telling it to display on the screen all the characters between the quotes, in this case the word of greeting. You can use the PRINT statement to tell the computer to display any message you wish.

You may ask what would happen if you don't spell the word PRINT correctly. Suppose you have typed PRONT instead of the correct word PRINT. The computer faithfully displays what you typed in at the bottom of the screen. The computer would not execute what you ask it to do until you press the key ENTER.

Now you press the key ENTER. This time the computer gives you a buzzer sound and inserts a blinking letter E after the word PRINT. The blinking letter E is called the error flag. This means that the computer cannot understand what you ask it to do. You can delete the whole statement by pressing the key BREAK (shifted space) and then retype the current statement. Pressing the key BREAK will clear what you have just typed in. Actually you need not retype the whole statement. You may just correct the letter "O" in the word PRONT to "I" to get the right word. First what you have to do is to move the blinking cursor just at the right of the word. Though you do not see the blinking cursor when the computer displays the error flag, the cursor is at the end of the statement. To move the cursor, press the key ← and → (shifted B and shifted N). ← move the cursor left one space and → move it right one space. Press the key ← 9 times will move the blinking cursor just at the right of the letter "O". Press the key DELETE (shifted .) will erase the letter "O". Delete will delete any one character just at the left of the cursor. Now insert the correct letter "I" by pressing the key I.

Since the computer do not allow the cursor inside a keyword, so you have to use the → key to move the cursor to the right of the word PRINT. Press the key → 2 times to move the cursor after the letter T and then press ENTER. The computer print the word "HELLO" at the top of the screen and the report message "OK" at the bottom.

Now try the statement

```
PRINT "100"
```

The computer obediently prints the number 100 on the top as expected. But type

```
PRINT 100
```

The computer again prints the number, without any fuss or error message about the missing quotation marks. In fact, the computer will let you PRINT any number at all without enclosing it in quotes.

Without further study, the computer can be used as a simple-minded desk calculator.

Try this on your computer

```
PRINT 2 + 3
```

The answer 5, appears on the top. The computer can do six different elementary arithmetic operations:

1. ADDITION – Indicated by the usual plus sign (+).

2. SUBTRACTION – Uses the conventional minus sign (–).

3. MULTIPLICATION – Many people use an "X" to represent multiplication. This could be confused with the letter "X". Some people use a dot (.), but this could be asterisk (*). To find 6 times 7 (in case you don't remember the answer), just type

```
PRINT 6 * 7
```

and have your memory jogged.

4. DIVISION – As is customary, use a slash (/). To divide 64 by 8, type

```
PRINT 64 / 8
```

and the correct answer will appear.

Try dividing 5 by 2. The answer is two and one half. The computer gives the answer to you in the decimal form 2•5.

One thing we should point out here is that you can do more than one arithmetic operation in the same instruction. For example, it is legal to say.

```
PRINT 1 + 2 + 3 + 4
```

The exact rules governing such usage will be given later, but you can experiment with it now if you wish.

6. EXPONENTATION – It is often handy to multiply a number by itself a given number of times. Instead of bothering to write

```
PRINT 2 * 2 * 2 * 2 * 2
```

You can substitute the shorthand

```
PRINT 2 * * 5
```

There is nothing special about exponentiation. It is just an abbreviation for repeated multiplication. In non computer-notation, this would be written with a superscript five, like this: 2^5

FORMAT FOR NUMBERS

Type

```
PRINT 12.340
```

Your computer responded with

```
12.34
```

and didn't PRINT the trailing zero. The computer does not PRINT leading or trailing zeros, that is, zeros that are at the beginning of a number and to the left of the decimal, zeros that are at the end of a number and to the right of the decimal.

Very, very small numbers (between about .000000000000000000000000000000000000003 and −.000000000000000000000000000000000000003) will be converted to zero by the computer (We hope that was the right number of zeros.) An easier way to write these numbers is 3 * 10 * * − 39. Don't take our word for it. Try it yourself.

Now type

```
PRINT 123456.789
```

Surprise! The last one digit is lost, and the number left behind is the closest approximation the computer can think of. This process is called "rounding". Try typing

```
PRINT 987.6543
```

Your computer did not round the number, but PRINTed it just the way you typed it. Madness you say? Ah, but there is a method to this seeming madness. Numbers are rounded only if they have more than eight digits. Any number that has fewer than nine digits will not be rounded. The computer does the best it can, but it only has eight digits to work with.

If you type a PRINT statement with a long number like

12345678000000

The computer responds with

1.2345678E+13

The number 12345678000000 and 1.2345678E+13 have the same value. Really the number PRINTed by your computer is in "scientific notation". If you need numbers like this you probably know how to read them.

Try some more numbers. How many digits can a number without a decimal point have before the computer changes it to scientific notation? If scientific notation seems complicated, don't worry. You probably won't be wanting to use numbers that require it for some time yet. Remember that any number will be PRINTed just the way you type it if the number is surrounded by quotes. However, the computer can't use numbers in quotes for arithmetic operations.

MORE ABOUT ENTER

So far, you have been pressing ENTER after every line, like a zombie. We thought we might tell you why this key gets so overworked. The reason is simple: without the ENTER, the computer does not know when you have completed the instruction. For example, you might start typing

```
PRINT 1 + 2
```

If the computer immediately jumped in and printed a 3, you might be upset because you had planned to type

```
PRINT 1 + 2 + 3
```

which would have given a different answer entirely. Since the computer can't tell when you have finished typing an instruction, you must tell the computer. You do this by pressing the ENTER key. Since you always have to do this after typing an instruction, we have (as you know) stopped mentioning ENTER after every instruction. Pressing ENTER after each instruction should be a habit by now, if you have been doing all the examples.

INTRODUCTION TO VARIABLES

On many simple calculators you can save a number for later reference or use. To do this, you put the number into a special place in the calculator – a place we shall call, for now, a pigeonhole. Usually this is done by pressing a key marked "M" for "Memory". On your computer you can do the same thing. For instance, to save the value 10, you type

```
M = 10
```

The value, 10, is not printed, just stored in the pigeonhole calling M. If you now type

```
PRINT M
```

the computer will print the value of M. Try typing the two statements. Now type

```
M = 20
```

and PRINT the value of M. It is 20, right? What happened to the 10? It is gone forever. The pigeonhole can hold only one value at a time. When you put a new value in M, the old value is erased. Type

```
PRINT "M"
```

What happens? There is a big difference between

M

and

"M"

It is just like the difference between these two statements in English:

MICE HAVE FOUR FEET.

"MICE" HAS FOUR LETTERS.

In one case we are referring to little furry things with long tails. In the other case we are referring to the word itself. This is how quotes are used in computers. When we say

```
PRINT "M"
```

we mean to print the letter itself. When we say

```
PRINT M
```

we mean to print what the letter stands for. You would never confuse the name of someone you love with the actual person that name stands for.

You can store the result of a computation in a pigeonhole. For example:

```
M = 4 + 5
```

You can see that the answer has been stored by PRINTing the value of M.

You can also use the value of M in further computation. For example, try this on your computer.

```
PRINT M + 2
```

Is the answer what you expected? Try some other calculations using M.

A simple calculator has one pigeonhole. Computers have hundreds of pigeonholes. The format term for pigeonholes is variables. But this term is somewhat misleading since pigeonholes don't behave like "variables" in mathematics. They are much simpler. Each one is merely a place where one value is stored.

A pigeonhole, or variable, can have almost any name that you like, so long as it starts with a letter. For example:

```
SUM = 56 + 34 + 1523 + 8
GAMEPOINTS = 45
PLAYER2 = 9
```

Some names are not allowed because they include a word that has a special meaning to your computer. These are known as reserved words. One of these words is "PRINT". Thus a variable's name must not have the word "PRINT" in it. Try typing

```
PRINTER = 6
```

or

```
FINGERPRINT = 9
```

All you get for your pains is an error message. Whenever a variable name gives you the error flag, it means that you have unwittingly included a reserved word in the name. Don't worry. Just choose another name.

When you are choosing names, make them reflect the use to which they are being put. This will make them easier to remember. Here is a useful trick. Let's say that you had some value in the variable PRICE, and you wanted to increase this value by 5. One way you could do this would be to PRINT the value of PRICE, then add 5 to that value, and finally store the resulting value back in PRICE. For instance:

```
PRICE = 28
PRINT PRICE
PRINT 28 + 5
PRICE = 33
```

But see how much easier it is to type

```
PRICE = 28
PRICE = PRICE + 5
```

Try the following statements in order:

```
PRICE = 2
PRINT PRICE
PRICE = PRICE + 3
PRINT PRICE
PRICE = PRICE * 6
PRINT PRICE
PRICE = PRICE / 10
PRINT PRICE
```

At the end of this sequence of statements, you will properly have the value 3. Is this correct? Is this what you expected? Try this sequence:

```
APPLES = 55
BANANAS = 11
QUOTIENT = APPLES / BANANAS
PRINT QUOTIENT
```

First think what answer you expect, then see if you are right. If you are not, find out why. Lastly, try these statements:

```
HELLO = 128
PRINT "HELLO"
HELLO = HELLO / 2
PRINT "HELLO"
HELLO = HELLO / 2
PRINT HELLO
```

What did you expect? What did you get?

PRECEDENCE, OR WHO'S ON FIRST

At certain old-fashioned banquets, the people were served their food according to a strict plan: first the guest of honor, then the female guests (in order of the rank of their husbands), then the male guests (in order of rank), and finally the host. No matter where they were seated, the waiter went among them choosing the appropriate persons to be served next. We could say there was a certain precedence among the diners. In a simple calculation like

```
PRINT 4 + 8 / 2
```

You can't tell whether the answer should be 6 or 8, until you know in which order (or precedence) to carry out the arithmetic. If you add the 4 to the 8, you get 12. If you then divide 12 by 2, you get 6. That's one possible answer. However, if you add 4 to eight-divided-by-two, you have 4 plus 4, or 8. This is another possible answer. Eight is the answer your computer will give. Here's how the computer chooses the order in which to do arithmetic:

1. When the minus sign is used to indicate a negative number, for example

```
-3 + 2
```

the computer will first apply the minus sign to its appropriate number or variable. Thus –3 + 2 evaluate to –1. If the computer did the addition first, –3 + 2 would evaluate to –5. But it doesn't. Another example is

```
BRIAN = 6
PRINT – BRIAN + 10
```

The answer is 4. (Notice, though, that in the expression 5 – 3 the minus sign is indicating subraction, not a negative number).

2. After applying all minus signs, the computer then does exponentiations. The expression

```
4 + 3 ** 2
```

is evaluated by squaring three (three times three is nine), and then adding four, for a grand total of 13. When there are a number of exponentiations, they are done from left to right, so that

```
2 ** 3 ** 2
```

is evaluated by multiplying 2 by itself three times (2 * 2 * 2) which is eight, and then multiplying that by itself (8). The answer is 64.

3. After all exponentiations have been calculated, all multiplications and divisions are done, from left to right. Arithmetic operators of equal precedence are always evaluated from left to right. Multiplication (*) and division (/) have equal precedence.

4. Lastly, all additions and subractions are done, from left to right. Addition (+) and subraction (–) have equal precedence.

Let's summarize the computer's order of precedence for carrying out mathematical operations:

First:	–	(minus signs used to indicate negative numbers)
Second:	**	(exponentiations, from left to right)
Third:	* /	(multiplications and divisions, from left to right)
Forth:	+ –	(additions and subtractions, from left to right)

Below, you will find some arithmetic expressions to evaluate. With each one, first do it in your head (or with the help of a hand-held calculator, or pencil and paper), and then try it on the computer. If your own answer is different from the computer's answer, try to find out why. We will give only the expressions here. You will have to put a PRINT in front of each one to get its value from the computer.

Unless you have a lot of experience with the way computers evaluate expressions, you should actually do these examples. Don't do them all at once and then check with the computer. Do an example by hand and then do it on the computer. Then go on to the next one, and so on.

```
3 + 2
4 + 6 – 2 + 1
8 * 4
4 ** 2 + 1
6 / 4 + 1
5 – 4 / 2
6 * –2 + 6 / 3 + 8
4 + –2
2 ** 2 ** 3 + 1
2 * 2 * 3 + 1
2 * 2 + 1 * 3
2 * 2 * 1 + 3
8 / 2 / 2 / 1
8 * 2 / 2 + 3 * 2 ** 2 * 1
20 / 2 * 5
```

No answer are given in this book. Your computer will give you the correct answers.

HOW TO AVOID PRECEDENCE

Suppose you want to divide 12 by four-plus-two. If you write

    12 / 4 + 2

You will get 12-divided-by-four, with two added on. But this is not what you wanted. To accomplish what you wanted in the first place, you can write

    12 / (4 + 2)

The parentheses modify the precedence. The rule the computer follows is simple: do what is in parentheses first. If there are parentheses within parentheses, do the innermost parentheses first. Here is an example:

    12 / (3 + (1 + 2) ** 2)

In this case doing the innermost parentheses, you first add 1 + 2. Now the expression is, effectively,

    12 / (3 + 3 ** 2)

But you know that 3 + 3 ** 2 is 3 + 9 or 12 so the expression has now been simplified to 12 / 12, which is one.

In a case like (9 + 4) * (1 + 2), where there is more than one set of parentheses, but they are not "nested" one inside the other, you just work from left to right. This expression becomes 13 * 3, or 39.

Here are some more expressions to evaluate. Again, if you are not familiar with computers, the few minutes you spend actually working these out and trying them on your computer will be very valuable. You will be well repaid for your efforts by being able to use the computer more effectively. Incidentally, most of these rules for precedence and parentheses hold good for most computer systems anywhere in the world.

    44 / (2 + 3)
    (44 / 2) + 2
    3 + (-2 * 2)
    (3 + -2) * 2
    100 / (200 / (1 * (9 - 5)))
    32 / (1 + (7 / 3) + (5 / 4))

CHAPTER 3

ELEMENTARY PROGRAMMING

Deffered execution

Up to now, when you typed

    PRINT 3 + 5 * 2

the computer would do what you told it to do, immediately. When a computer performs according to the statement you have given it, it is said to execute that statement. Thus, you have been using the computer to do immediate execution of each statement you have typed on the keyboard.

You are about to learn how to store statements for execution at a later time (deferred execution). Now type

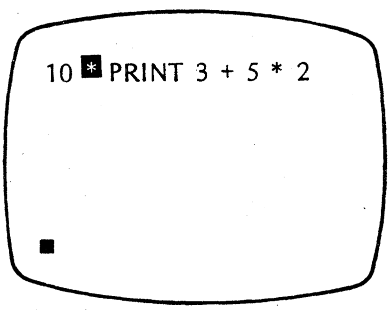
    NEW

The command NEW starts the BASIC system again, deleting any previous programs and variables.

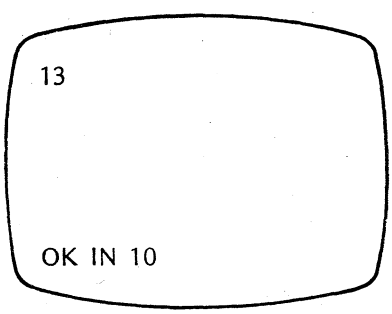
To tell the computer to store a statement, just type a number before typing the statement. For example, if you type

    10 PRINT 3 + 5 * 2

nothing seems to happen but the line you just type in enter to the top line of the screen.



The computer has stored the statement. To ask the computer to execute the statement, you must type RUN then press ENTER key. Try it. The answer 13 appears on the top line of the screen, and a remark "OK IN 10" at the bottom line which means statement 10 has been finished.



To see that your computer has stored the statement, you type the command

LIST

Try it.

The statement 10 PRINT 3 + 5 * 2 appears on the screen. Typing RUN caused your stored statement to be executed, but the computer has not forgotten the statement. You can RUN the same statement as many times as you like. Try it.

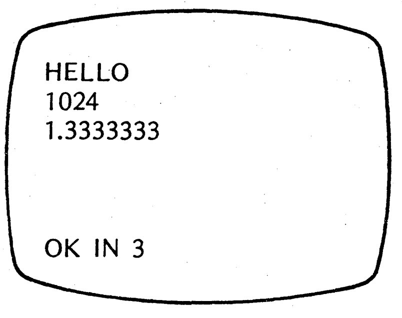
It is possible to store many statements by giving each of them a different number. Try typing this:

```
NEW
1 PRINT "HELLO"
2 PRINT 2 ** 10
3 PRINT 4 / 3
```

Nothing much has been happened so far. But now type

RUN

and watch the answers appear.



What's more, the computer does not forget the stored statement when you clear the screen. Here's a new way to clear the screen:

CLS

This command can be used in deferred execution as well as immediate execution.

To try this out type

100 CLS

Now when you type

RUN

the computer faithfully executes the stored statement and clears the screen. Type

NEW

and then

LIST

and see what happens. Typing NEW has caused the stored statement to be lost permanently. Type

RUN

and nothing appears on your screen. That is because your old statement has been erased by the NEW command.

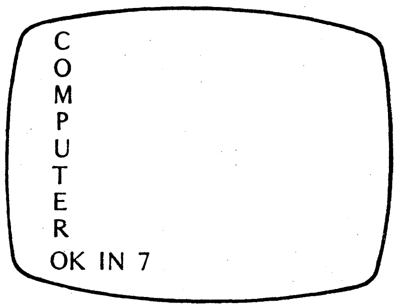
The numbers that we put in front of statements, in order to tell the computer to store them, are called line numbers. The computer stores and executes statements in order of increasing line number. To see this in action, erase the statements you stored by typing

NEW

and then type these statements:

```
2 PRINT "O"
3 PRINT "M"
1 PRINT "C"
7 PRINT "R"
6 PRINT "E"
5 PRINT "U"
4 PRINT "P"
```

Notice that zero is an allowed line number, but it refers to immediate mode execution. The highest line number that you can use is 9999. Now RUN these instructions. The results should look like this:



To see what has happened inside the computer, type

LIST

Notice that you do not have to LIST a set of instructions before you RUN them. It is, however, a good idea to do so.

A set of instructions that is executed when you type RUN is called a program.

The program was meant to print the word "COMPUTER" vertically. But, it seems, a PRINT statement was left out. How can you add it in? Only by retyping the statements with line numbers 6 and 7 as statements 7 and 8, and adding a new line number 6. To make the corrections type this:

```
6 PRINT "T"
7 PRINT "E"
8 PRINT "R"
```

Notice that in whatever order statements are entered, the computer stores them with their line numbers in numerically ascending order. Now RUN this program.

It is boring to retype these statements in order to merely add one in the middle. It is, therefore, good programming practice to leave some line number room between lines, and before the first line. Type

```
NEW
```

to eliminate that program and put in this one:

```
10 PRINT "G"
20 PRINT "D"
```

When you RUN this program it doesn't quite print the word "GOD" vertically. But now you can go back and type

```
15 PRINT "O"
```

LIST and RUN this program. From now on this book will start all programs at a reasonably high line numbers so there will be adequate room for inserting statements.

USING THE LINE NUMBER

One of the unique features of the computer is the automatic line number function. The key "LINE NO." is located above the X key. Pressing this key (By holding down the SHIFT key then press X key), will clear the bottom part of the screen and enter automatically a line number which is equal to ten plus the line number of the current line. It will be limited to 9999. Of course you can enter a line number manually by the numerical keys above the keyboard. If you are a programmer or you are going to develop your own programs, you will be found that this feature of automatic line number entering is very helpful.

Now, enter your own program by using the automatic line numbe key.

(1) Type NEW. Press LINE NO (shifted X). a 10 appears on the bottom part of the screen (without ENTER).
(2) Type PRINT "MICROCOMPUTER" and press ENTER, a complete line:

```
10■ PRINT "MICROCOMPUTER"
```

appears on the top line of the screen. The symbol ■ indicates that this line is currently entered.
(3) Type LINE NO (without ENTER) again, you will automatically get a new line number 20 appears on the bottom part of the screen.
(4) Type PRINT "YOUR GOOD COMPANION" follow to the 20, then ENTER, you will get

```
10 PRINT "MICROCOMPUTER"
20■ PRINT "YOUR GOOD COMPANION"
```

on the screen. As you can see, the symbol ■ moves to the current line 20. You can use the LINE NO to get lines 30, 40 ..... and so on

```
30 PRINT "PROGRAM BY JOHN"
40 PRINT "ON 1/5/1983, N.Y."
```

Now you get the completed program as below:

```
10 PRINT "MICROCOMPUTER"
20 PRINT "YOUR GOOD COMPANION"
30 PRINT "PROGRAM BY JOHN"
40 PRINT "ON 1/5/1983, N.Y."
```

Try to RUN it yourself.

EDITING PROGRAMS

You see that a symbol ⧆ followed the line number 40? This is called the Program Cursor, and the line it points to is the current line. You can move this cursor up to line 30 by the key ↑ (shifted C) and down to line 40 again by press the key ↓ (shifted V).

Now suppose you want to change some thing in line 30. Use the key to move the program cursor to line 30, then press EDIT key (shifted M), and a copy of line 30 will be displayed at the bottom line of the screen.

```
30 PRINT "PROGRAM BY JOHN"
```

Don't forget other functional keys ← (shifted B), → (shifted N) and DELETE (shifted .) which we have learnt before. Now move the blinking cursor to the left of JOHN. Type L., the statement appears as

```
30 PRINT "PROGRAM BY JOHN L."
```

Now press ENTER, then the new line 30 will replace the old one in your program.

Now, RUN the program to see what have been changed.

You can also change a line by re-type it with correction. For example, type in

```
20 PRINT "YOUR BEST COMPANION."
```

and press the ENTER key. This new line will replace the old line 20. Try it.

To delete any line in your program, just type the line number and press ENTER. For example, type 40 then ENTER, the last line in your program will disappear.

If you are about to fix up line 40 so you type

```
40
```

and think about it and decide not to change the line after all. Don't press ENTER key. Either use DELETE key to erase the line number, or use the BREAK key (shifted SPACE) to abort the command.

MORE ABOUT LISTING PROGRAMS

Anytime the computer is not executing, you can check your program by using the LIST command. Type LIST and press ENTER, and you will see the program stored in your computer. (But notice that the program cursor disappears from the lines, it points to line number 0).

Now type

```
LIST 30
```

Notice that only line 30 and 40 appear on the screen, and the program cursor now is at line 30.

The LIST command starts listing your program from whatever line is given to the end of the program, if possible. If you use the LIST command without a line number, the listing will show in sequence started from the line number 0. That is why when you LIST the program, the program cursor do disappear, it goes to the "line 0".

Note that the screen can only show 22 lines at a time. If your program is longer than 22 lines, you will have to list it in stages.

A WORD ABOUT LEARNING BASIC LANGUAGE

Many times there are questions you can ask about the BASIC language that are not answered directly in this book. For instance, in the statement

```
PRINT "HELLO"
```

do you have to put a space after the word PRINT? Rather than give you the answer, we recommend that you simply turn to your computer and try it both ways. Usually a simple experiment will answer your question and, since you have taken the time to try it yourself, you will remember it far better than if you had merely read it.

Unconditional Branch – GOTO statement

At this point, you are begining to fly, so this section will discuss "loops". Type the following lines (after typing NEW, to erase any old programs that may be around):

```
10 PRINT "HELLO"
20 GOTO 10
```

Line 10 of this program PRINTs the message "HELLO". Line 20 does just what it seems to say: it causes program execution to go to line 10. What happens then? The program PRINTs the message "HELLO" again, then it executes line 20, which say to do line 10 over again, and so on. Forever, this is a "loop".

A loop is a program structure that exists when the program includes a command to return to a statement executed previously. Now

```
RUN
```

the program. The computer will print "HELLO" continuously until the screen full.

Try the commands

```
RUN 10
RUN 20
```

instead of RUN. You will notice that there is no distinction on the screen display. Then, what is the distinction between these RUN commands?

The command RUN without any line number tells the computer to execute the program starting from line 0, where RUN 10, RUN 20 tells the computer to execute the program starting from line 10 and line 20 respectively.

There is a question. What is the difference between RUN 10 and GOTO 10? One big difference between RUN and GOTO is that GOTO does not clear the value assigned to the variable. For example, if you RUN the program below

```
10 LET A = 5
20 PRINT A
```

Now enter a LET command

```
LET A = 10
```

then type

```
GOTO 20
```

and answer 10 print for A. But if you type:

```
RUN 20
```

an error message "UV IN 20" will be appeared, because RUN clears the variable memory areas, hence value of variable A is undefined.

THE TRUTH

The computer can distinguish between what is true and what is false. Since this is more than most of us can do, a few words of explanation are in order. The symbol > means "greater than", the assertion 2 > 1 (which is read "two is greater than one") is certainly true. The computer uses the number 1 to indicate truth.

If you type

```
PRINT 2 > 1
```

the computer will reply with a one. The assertion 3 > 4 is false. The computer uses the number 0 to indicate falsehood. If you type

```
PRINT 3 > 4
```

the computer will reply with a zero.

The symbol < means "less than", and you can make statements using it as well. Here is the full set of symbols used in making assertions:

>	greater than
<	less than
=	equal to
>=	greater than or equal to
<=	less than or equal to
<>	not equal to

To type the symbols for "greater than or equal to" and "less than or eaual to" on your keyboard, you must first type either a <or a > and then type an =. To type symbol for "not equal to" you must type a < and then a >.

Think about and then test to see which of these assertions are true, and which are false.

```
5 <> 5
6 > 2
8 <= 8
734 = 532
5 < 9
54 >= -6
-9 < -2
-5 >= -8
8 <> -8
```

Assertions can include variables and expressions as well as numbers. For example

```
PRINT (45 * 6) <> (45 + 6)
```

will print the value 1 since 270 is not equal to 51 (remember that 1 means the assertion is true).

You have seen that the computer can tell truth from falsehood in simple assertions about numbers. However, an assertions such as ABLE > BAKER may be true or false, depending on the value of the two variables, ABLE and BAKER. If

ABLE = 5
and
BAKER = 9
then the assertion
ABLE > BAKER
is false. But if
ABLE = –8
and BAKER = –15
then the assertion
ABLE > BAKER
is true.

Assertions have the numerical values of zero or one. They can be used in arithmetic expressions instead of ones and zeros. For example,

```
PRINT 3 + (4 > 2)
```

will print the value 4. The statement

```
T = 4 <> 3
```

gives T the value 1, since 4 does not equal three, and thus the assertion 4 <> 3 has the value 1. The statement

```
HOT = 67 = 19
```

looks very confusing at first, but it is easily understood. Since 67 does not equal to 19, the assertion 67 = 19 is false and has the value zero. The value of 0 is given to the variable "HOT".

As we have seen, the computer uses 1 to mean true, and 0 to mean false. If something is not true, it is false. If something is not false, it is true. This may not always be the case in real life, but it is always the case wich computers. Try this on the computer:

```
PRINT NOT 1
```

and then try

```
PRINT NOT 0
```

The computer agrees: not true is false and not false is true. Of course, you can use expressions instead of ones and zeros. For example

```
PRINT NOT (45 > 3)
```

The sentence
TRIANGLES HAVE THREE SIDES.
is true. And the sentence
THIS BOOK IS IN ENGLISH.
is true. Consider the sentence
TRIANGLES HAVE THREE SIDES AND THIS BOOK IS IN ENGLISH.
Is this sentence true or false? It is true. Consider the sentence TRIANGLES HAVE FIVE SIDES AND THIS BOOK IS IN ENGLISH.
This sentence, as a whole, is false. Lastly, consider the sentence TRIANGLES HAVE FIVE SIDES AND THIS BOOK IS IN CHINESE.

This sentence is also false. In general, when you combine two sentences, or assertions, by joining them with the word AND, you find that

a. The new sentence is true if both original sentences were true.
b. The new sentence is false if at least one of the original sentences was false.

The computer knows how to determine whether an assertion containing the connecting word AND is true or false. Test your computer with the following instructions; try to predict each answer:

```
PRINT 1 AND 1
PRINT 1 AND 0
PRINT 0 AND 1
PRINT 0 AND 0
PRINT (3 > 2) AND 0
PRINT (NOT 0) AND (1 = 2)
```

Is this sentence true or false?
A TRIANGLE HAS THREE SIDES OR THIS BOOK IS IN LATIN.
It's true. A triangle does have three sides, even if this book isn't in Latin, so the sentence as a whole is true. In general, when you combine two sentences by joining them with the word OR, you find that

a. The new sentence is true if one or both of the original sentences were true.
b. The new sentence is false if both of the original sentences were false.

The computer can also determines if an assertion containing OR is true or false. Try each of these on your computer – after figuring out what the answer should be.

```
PRINT 1 OR 1
PRINT 1 OR 0
PRINT 0 OR 1
PRINT 0 OR 0
PRINT (4 <> 5) OR (4 = 5)
PRINT 1 OR (0 AND 1)
PRINT ((3 > 4) OR (50 < 100)) AND (NOT 0)
```

AND, OR, and NOT will become very useful in the next section.

You have already found that in the statement

```
PRINT 1 OR 0
```

the computer regards 1 as true and 0 as false. Now try this:

```
PRINT 23 OR 0
```

and this:

```
PRINT -123 AND 45.6789
```

In assertions, the computer regards not only 1, but any number which is not zero, as true. However, when the computer figures out the value of an assertion, that value will always be either 0 or 1.

While the following box gives the precedence rules for AND, OR, and NOT, we stringly recommend that you use parentheses to make your statements clear.

ORDER OR PRECEDENCE
FOR OPERATIONS USED
SO FAR IN THIS TEXT:

1. ( )
2. NOT — (for negative values)
3. **
4. * /
5. + —
6. > < = >= <= <>
7. AND
8. OR

THE IF STATEMENT

Suppose you want to print out integers from 1 through 10, one number on a line. An obvious way to do this is

```
NEW
210 PRINT 1
220 PRINT 2
230 PRINT 3
```

And so on. But this would require 10 statements, and if you wanted to print the integers from 1 through 200 this way, it would require 200 statements. Using what you have already learned, you can PRINT integers from 1 on up, in just four statements by using a loop:

```
200 N = 1
210 PRINT N
220 N = N + 1
230 GOTO 210
```

There is a way to control how long a loop runs. What you want is a statement that does a GOTO if N is greater than 11. The answer to your wishes is the IF statement. If a condition is met, the computer will skip the GOTO instruction and execute the instruction on the next line. If there is no next line, the program will end.

Here is a program that counts from 1 to 10 and then stops:

```
200 N = 1
210 PRINT N
220 N = N + 1
230 IF N <= 10 THEN GOTO 210
```

In general, the IF statement works like this:

```
IF arithmetic expression THEN any statement
```

First, the arithmetic expression is evaluated. If it evaluates to zero (false) all the rest of that program line is ignored, and the computer goes on to the next line. If the arithmetic expression is not zero (true) the remaining portion of that program line is executed.

The IF statement is a very powerful one, and it will appear in almost every program you write.

FOR/NEXT LOOPS

Loops, when executed by computer programs, have a top and a bottom. In the program

```
NEW
100 N = 0
110 PRINT N
120 N = N + 1
130 IF N <= 12 THEN GOTO 110
```

line 110 is the top of the loop, and 130 is the bottom. The program prints the integers from 0 to 12 inclusive. The number 12 is the limit of the loop. Another way to write a loop is to use a statement we have not discussed yet: the FOR statement. We can use this statement to rewrite the previous program.

```
200 FOR N = 0 TO 12
210 PRINT N
220 NEXT N
```

Notice the control variable N must be a single letter.

Type

```
RUN 200
```

to execute this program. If you just type RUN, the program at line 100 (the lowest line number around) will be executed.

Line 200 contains the new FOR statement. It starts by setting N to the value 0. This is exactly the same task that line 100 performed. Then line 210 is executed. The bottom of the FOR loop is in line 220. The variable N is increased by 1 and then compared to the upper limit specified in the FOR statement: 12. If N is not over the limit, execution continues at the statement immediately following the FOR. If the variable is over the limit, the program drops through (out of the loop) to the statement after the NEXT. In this case, the program drops through and, not finding any more lines, terminates the program.

The most obvious advantage of the FOR/NEXT method of constructing loops is that it saves a statement. The most important advantage is that you don't have to think so hard when writing a loop if you use a FOR/NEXT loop. If you wanted to draw a horizontal line on the middle of the screen, you could type

```
10 N = 21
20 FOR M = 0 TO 63
30 PLOT M, N
40 NEXT M
```

Another advantage is that it is much easier to read a single FOR statement than to look through three statements to figure out what a loop is doing. To find the bottom of a FOR/NEXT loop, all you have to do is look for a NEXT which has the same variable as the FOR.

It might be well to mention that, although you should know how the FOR statement works, you don't have to use it. It doesn't add any new abilities to those you already have. It just makes some programs easier to write (for some people).

To PRINT just the even numbers from 0 to 12, you could use the program

```
100 E = 0
110 PRINT E
120 E = E + 2
130 IF E <= 12 THEN GOTO 110
```

The secret is in line 120, where 2 is added to E. We say that the loop steps by two. To step by two in a FOR loop, you would type

```
200 FOR E = 0 TO 12 STEP 2
```

The rest of the program would look like lines 210 through 220 on the previous page except that the name N would have to be changed, wherever it occurs, to the name E. Try it. The STEP may be any number in the range of the computer. It can even STEP backward, for example

```
200 FOR E = 39 TO 15 STEP −3
```

Type this line and try it by typing

```
RUN 200
```

You should play with the FOR statement for a while, if you wish to learn to use it. A number of the example programs from this point on will use the FOR statement.

Along with the convenience of the FOR statement come some limitations. For example, FOR/NEXT loops may be nested, but may not cross. Here are a few examples which demonstrate the idea.

```
NEW
10 FOR C = 1 TO 11
20 FOR M = 1 TO 16
30 PRINT "▀ ▄ ";
40 NEXT M
50 FOR N = 1 TO 32
60 PRINT "▚ ";
70 NEXT N
80 NEXT C
```

This program is an example of two-level nesting. Think about it and RUN this program before going on to the next. Remember, when writing programs using FOR statements, each FOR must have a matching NEXT.

A WRONG PROGRAM

```
NEW
500 FOR N = 10 TO 20
510 PRINT N
520 FOR J = 30 TO 40
530 PRINT J
540 NEXT N
550 NEXT J
```

This program won't work. Its loops are crossed, which not only gives an error message, but doesn't make any sense. Whenever you find yourself writing crossed loops, your thinking has gotten tangled. If you are sure you know what you are doing, and still want to cross loops, use loops made with IF statements. You can cross those all you want, for what good it will do for you.

PRINTS CHARMING

As an experiment, type this program and see what it does when you RUN it.

```
NEW
100 PRINT "HELLO"
110 GOTO 100
```

Stop the program with BREAK. Then change line 100 by just one symbol: comma (,)

```
100 PRINT "HELLO",
```

and RUN the program again. As you can see, this PRINTs the word in columns. Now substitute a semicolon (;) for the comma (,)

```
100 PRINT "HELLO",
```

and RUN the program again. This time the output is packed. This means that there are no spaces between what you told the computer to PRINT. It prints HELLO after HELLO, until the screen is quickly filled.

Change the program by adding this statement

```
90 N = 88
```

and changing line 100 to read

```
10Q PRINT N
```

RUN this program. Now change line 100 to

```
100 PRINT N,
```

and RUN it again. Then change line 100 to

```
100 PRINT N;
```

and observe that the comma and semicolon can also be used with numerical values. The ability to place numbers one after the other without intervening spaces is sometimes quite useful. Commas and semicolons can be used within a PRINT statement. Clear the old program with NEW, and type

```
100 STRIKES = 2
110 BALLS = 3
120 PRINT STRIKES, BALLS
```

You can make clearer output by including messages in the PRINT statement. For example, change line 120 into

```
120 PRINT "THE STRIKES AND BALLS ARE", STRIKES, BALLS
```

Notice that you probably want to have a space after the word ARE lest the number of strikes gets printed too close to it. If you don't think that the large space between the number of strikes and balls looks nice, you could use the statement

```
120 PRINT "THE STRIKES AND BALLS ARE    "; STRIKES; "      "; BALLS
```

In this version, a blank is put between the numbers of strikes and balls. Perhaps the pretties way of doing this (are you trying all of these on your computer?) is

```
120 PRINT "STRIKES    "; STRIKES; "      BALLS      "; BALLS
```

This gives you a scoreboard-like display.

PRINTING WITH TAB & AT

Let's say that you wanted to PRINT the word HERE starting in the 10th column (the screen is 32 columns across, by the way), you could use this statement

```
120 PRINT "          HERE"
```

(You have to take our word for it that there are ten blanks before the word HERE). Or you could use the TAB feature. Just as on typewriters, you can set a tab on the computer. The statement

```
120 PRINT TAB (10); "HERE"
```

has the same effect as putting 9 blanks in the quotes, as we did above. Try it, you'll like it.

By combining the TAB with the FOR loop you can program some nice visual effects. For example:

```
NEW
200 FOR N = 1 TO 22
210 PRINT TAB(N); "X"
220 NEXT N
```

There are 22 (not 32) horizontal printing lines. That, by the way, is why the upper limit in the loop in the program above is 22. TAB cannot be used to move backwards (to the left) on a line. Only forward moves are carried out.

To print on a particular position, you can use the PRINT AT command. Try

```
PRINT AT 0,0; "T"          (print at top left)
PRINT AT 0,31; "T"         (print at top right)
PRINT AT 21,0; "B"         (print at bottom left)
PRINT AT 21,31; "B"        (print at bottom right)
```

For use with the PRINT command, the screen is divided into 22 rows and 32 columns. The rows are numbered from 0 to 21, from the top to bottom; and the columns from 0 to 31 from left to right. As you can see from the above examples, the first number after AT is the row number, and the second one is the column number. That statement has the following form

```
PRINT AT row, column; item
```

and item can be either a number or a string. For example, type

```
PRINT AT 5,5; "THE COMPUTER"
```

If you input a column number over the range limited, you will get an error message on the screen. To try this, you type

```
PRINT AT 5,32; "HELLO"
```

You will get IR message appears on the screen. IR means "integer out of range". The reason is that 32 in the PRINT AT statement is out of the allowed range. (0 to 31)

If you input a row number over the range limited (0 to 21), you will get the report SF message appears on the screen. Try this

```
PRINT AT 23,5; "HELLO"
```

The reason is that 23 in the PRINT AT statement is out of the allowed print range.

Here is a program that demonstrates the use of PRINT AT:

```
NEW
600 FOR X = 0 TO 21
610 FOR Y = 0 TO X
620 PRINT AT X,Y; "COMPUTER"
630 NEXT Y
640 NEXT X
650 GOTO 600
```

Before you RUN this program, try (it ain't easy!) to figure out what it will do. It's both surprising and pretty.

More computer programming

Statements: REM, INPUT, CONT, STOP, SCROLL, PAUSE, INKEY$

Type in this program

```
NEW
10 REM THIS PROGRAM EXTRACTS SQUARE
20 INPUT A
30 PRINT A, A*A
40 GOTO 20
```

Now RUN it. Apparently the screen goes blank, and nothing happens, but the blinking cursor appears in the bottom left-hand corner: the machine has gone into input mode. This is the effect of the INPUT statement in line 10. The machine now is waiting for you to type in a number (or even an expression).

Just a minute though, what happened to line 10? REM in line 10 stands for remark, or reminder, and is there solely to remind you of what the program does. A REM statement consists of REM followed by anything you like, and the computer will ignore it.

Now type in some number, 4. For example, then ENTER, the 4 and its square appear on the screen, and again the computer do want another number. This is because of line 40, GOTO 20. The computer jumps back to line 20 and starts again.

To stop it, press the key BREAK (shifted SPACE), the computer reports the message "BK IN 20". BK stands for BREAK. If you want more the square of a number, then type:

```
CONT
```

(CONT stands for CONTINUE) and the computer will clear the screen and ask for another number.

Notice that the screen is cleared not because the CONT statement, but because it is a command. All commands clear the screen first.

Keep entering numbers until the screen is full and you get the report "SF IN 30". If you want to continue entering numbers, you must type CONT. To avoid the screen full, add the statement

```
15 SCROLL
```

Now, when you RUN this program, the output will appear in the bottom line of the screen and scroll upward.

SCROLL moves the whole display up one line, with losing the top line and sets the print position to the beginning of the bottom line.

THE PAUSE

The PAUSE command stops execution and refreshes the display only. PAUSE has the following form:

```
PAUSE N
```

where N is in units of one TV frame. Two frames make up a complete picture on your screen. The image for a standard U.S. TV-transmission is formed at the rate of 1/60 of a second; therefore, the smallest unit for a PAUSE is 1/60 of a second. The maximum number of PAUSE allowed is 65,535. When N > 32768, the computer halts until a key on the keyboard is pressed down.

Let us try timing a PAUSE. Since one frame = 1/60 second, 20 seconds should equal to 1,200 frames. Type

```
PAUSE 1200
```

Notice that everything disappears from the screen. Keep your PAUSE time from when you press the ENTER key until a OK report appears.

The problem is that if using a PAUSE in your program that other statements will be collected the time. For example, type

```
10 N = 0
20 PAUSE 60
30 PRINT AT 0,0; N
40 N = N + 1
50 GOTO 20
```

This program acts as a digital timer. About every second, it prints the elapsed time in seconds since the program started. Although a PAUSE lasts one second, the computer takes some time to execute the other statements. To compensate for the difference, you have to adjust the PAUSE 60 downward. Unfortunately, the smallest increment you can adjust is 1/60 seconds, this is not accurate enough. Another problem is the annoying screen flicker that may occurs when it reactivates the screen.

READING THE KEYBOARD

The input command is useful for calculations or other data processing in which you want the computer to stop, wait for input,. then continue execution. However, in designing games, you may want the computer to respond for input but without stopping the program. A familiar example is the joystick of a computer game. The game machine continues to response as you move the joystick, instead of halting the action to ask for input.

Your computer has a special statement called INKEY$, which can be used to tell automatically what key is pressed. If a key is pressed, the INKEY$ function returns either its character or the null string "   " — means no character. Now run the program below:

```
10 K$ = INKEY$
20 PRINT K$;
30 GOTO 10
```

An ever shorter version is

```
10 PRINT INKEY$;
20 GOTO 10
```

and a shorter version still is

```
10 PRINT INKEY$;
20 RUN
```

Notice how quickly INKEY$ scans the keyboard.

DRAWING PICTURES BY JOYSTICK

You can easily draw pictures on the screen by using the commands we have discussed so far. Try this program to draw a vertical line of asterisk.

```
10 X = 11
20 Y = 16
30 K$ = INKEY$
40 IF K$ = "R" THEN X = X + 1
50 IF K$ = "4" THEN X = X – 1
60 IF K$ = "F" THEN Y = Y + 1
70 IF K$ = "7" THEN Y = Y – 1
80 PRINT AT X,Y; "*"
90 GOTO 30
```

You can move the asterisk all over the screen and draw all kinds of patterns by your joystick. Of course, you can substitute any character for the asterisk.

RUN it and you will find that you can move the asterisk all over the screen and draw all kinds of patterns.

SUBROUTINES

Suppose you want to draw a square, centre at (31,21), the distance from centre to side S = 4 units. Here is a program to draw such square:

```
NEW
1000 REM PROGRAM TO DRAW A SQUARE
1010 REM S: CENTRE TO SIDE DISTANCE
1020 S = 4
1030 FOR L = –S TO S
1040 PLOT 31 + L, 21 – S
1050 PLOT 31 + S, 21 + L
1060 PLOT 31 – L, 21 + S
1070 PLOT 31 – S, 21 – L
1080 NEXT L
```

There is nothing wrong with this program, it does draw a square with S=4 units. Now suppose you want to draw another square on the screen. You could rewrite this program with new values for centre co-ordinates, but that is a bother. There should be some way of using the same program to draw a square anywhere on the screen without having to rewrite it each time.

The key to doing this begins with the concept that you can assign the co-ordinates of the centre to be variables (X,Y). Any time you want to draw the square on a new place, you just need to assign new values to (X,Y).

With these facts in mind, you can rewrite your program to "centre" the square at almost any point (X,Y) on the screen. Why "almost" any point? Because, if you choose a centre point at an edge of the screen, the square will go off the screen, and this might give you an "IR IN 1040" (or some other line number) message. Here is an improved program.

```
1000 REM PROGRAM TO DRAW SQUARE
1010 REM (X,Y) = CENTRE S = SIDE
1020 S = 4
1030 FOR L = -S TO S
1040 PLOT X + L, Y - S
1050 PLOT X + S, Y + L
1060 PLOT X - L, Y + S
1070 PLOT X - S, Y - L
1080 NEXT L
```

This program can't be run just as it is. First, you must choose the values for X and Y. Type:

```
10 REM FIRST SQUARE CENTRE
20 X = 31
30 Y = 21
```

If you try to RUN this, you do get a square at the desired location, but the program ends there. We want to draw two square on the screen, what if you could write

```
40  EXECUTE 1000 TO 1080
50 REM SECOND SQUARE CENTRE
60 X = 48
70 Y = 12
80  EXECUTE 1000 TO 1080
```

do the portion of the program at line 1000 use again and then end.

Would't that be nice and easy? You know that the computer can't read those strange instructions at lines 40 and 80. It can, however, read

```
GOSUB 1000
```

in computer. A program such as the one starting at line 1000 is called a subroutine. GOSUB 1000 tells the computer to GOto the SUBroutine beginning at line 1000 and start executing at that statement. It also tells the computer to come back to the line that follows the GOSUB statement when it is finished with the subroutine. The computer knows the subroutine is finished when it encounters a RETURN statement. To make your square-drawing partial-program into a complete subroutine, add the line

```
1090 RETURN
```

Now you can write that "what if you only could" program:

```
10 REM FIRST SQUARE CENTER
20 X = 31
30 Y = 21
40 GOSUB 1000
50 REM SECOND SQUARE CENTRE
60 X = 48
70 Y = 12
80 GOSUB 1000
```

Now RUN the program. You get an error message:

"RG IN 1090"

but otherwise the program seems to RUN fine. In effect, you have added a new statement to the program: a square-drawing statement. Now you can use the statement

```
GOSUB 1000
```

to draw one of these special squares at whatever X,Y location you have chosen.

The portion of the program from line 1000 to 1090 is called a subroutine or subprogram. The portion of the program from line 20 to line 80 is called the main program.

To see the program's flow, we trace the program from the smallest line number. It is 10. Here the program begins at line 10, continues through the main program until the subroutine call, then executes the subroutine, goes back to the main program, executes the subroutine again, and, not finding any smaller line numbers, goes to line 1000 and executes the subroutine again. This is where the problem occurs. Do you understand the error message now?

To remedy the problem, add this new line to the program:

```
90 STOP
```

A STOP statement stops the execution of the program, giving report "ST IN 90". RUN the program once more. No more error message!

Up to now, we can control the location of the square in a easy way. Can we use the same method to vary the size of the square? Of course, we can. Add these lines.

```
35 N = 6
75 N = 3
1020 S = N
```

RUN and observe the result.

Here is an improved program to draw squares with different size. Type:

```
10 REM CHOOSE CENTRE AND MAX. SIZE
20 X = 31
30 Y = 21
40 M = 10
50 REM PROGRAM TO DRAW CONCENTRIC SQUARE
60 FOR N = 1 TO M + M STEP 2
70 GOSUB 1000
80 NEXT N
90 STOP
1000 REM PROGRAM TO DRAW SQUARE
1010 REM (X,Y) = CENTRE S = SIZE
1020 S = N
1030 FOR L = -S TO S
1040 PLOT X + L, Y - S
1050 PLOT X + S, Y + L
1060 PLOT X - L, Y + S
1070 PLOT X - S, Y - L
1080 NEXT L
1090 RETURN
```

CHAPTER 4

MATHEMATIC FUNCTIONS

Your computer has some math functions that you can find at the keyboard, they are:

SIN	(since)
COS	(cosine)
TAN	(tangent)
LOG	(natural logarithm)
SGN	(signum)
RND	(random)
ASN	(arcsine)
ACS	(arccosine)
ATN	(arctangent)
EXP	(natural exponent)
ABS	(absolute magnitude)

You can get these functions by holding down the SHIFT key and pressing the corresponding key.

EXP

The exponential function gives the exponential relative to the base of natural logarithm, that is e = 2.7182818.

```
PRINT EXP 2.5
```

gives the answer 12.182494

LOG

The natural log is the inverse function of the EXP. In general

```
X = LOG (EXP X)
```

Try the following

```
PRINT LOG 2.7182818
```

Notice that you cannot get back 1 exactly. That is because the number 2.7182818 is not the exact value of e. Now try

```
PRINT LOG EXP 1
```

You get the correct answer this time.

ABS

The absolute value function always returns the positive value of a number. Try

```
PRINT ABS 5
PRINT ABS –5
```

You get the answer 5 in both cases.

SGN

The signum function return a 1 if the number is positive, 0 if the number is 0, and −1 if the number is negative. Try

```
PRINT SGN 5
PRINT SGN 0
PRINT SGN −5
```

RND

The random function gives a number greater or equal to 0 and less than 1. RND is not a true random number generator. It takes a sequence of 65,536 numbers that are almost random, and is called pseudo-random generator. Try

```
PRINT RND
```

to see whether the result is random.

There is another function RAND related to RND. You can use RAND to start RND off at a definite place in the 65,536 number sequence. This is done by typing RAND followed by a number between 1 and 65,536. Try this

```
RAND 1
```

and then

```
PRINT RND
```

Type both of these several times. The answer from RND will always be 0.0022735596, not a very random sequence.

```
RAND 0
```

(or you can omit the 0) acts slightly different. It judges where to start RND off by how long the computer has been turned on.

TRIGONOMETRICAL FUNCTIONS

The SIN, COS, TAN, ASN, ACS, and ATN functions in your computer all work in radians, not degrees. Try

```
PRINT SIN 3
```

where the 3 is in radians but not in degrees. If you want to use degrees instead of radians, you must use the format

```
N = M*PI/180
```

where M is the degree value. To evaluate SIN 30, you must type

```
PRINT SIN (30*PI/180)
```

CHAPTER 5

GRAPHIC

PLOT AND UNPLOT

The screen you can display on has 22 lines and 32 columns, making 22*32 character positions, and each position has 4 pixels. Each of the characters is made up of 8*8 dots matrix and each pixel is a 4*4 dot matrix. A pixel is specified by two numbers, which is the cordinate of the pixel on the screen. The origin of the coordinate is defined at the bottom left hand corner of the screen. So the pixel (0,0) locates at the bottom left hand corner. The first coordinate of the pixel is how far it is across the screen. This number can range from 0 to 63. The second coordinate of the pixel is how far it is up from the bottom and the number can range from 0 to 43.

The statement

```
PLOT x,y
```

will plot a pixel at (x,y) and to erase it, use the statement

```
UNPLOT x,y
```

The following program plots and unplots random points on the screen

```
10 PLOT INT (RND*64), INT (RND*44)
20 UNPLOT INT (RND*64), INT (RND*44)
30 GOTO 10
```

The following program plots the graph of a SIN function for values between 0 and 2.

```
10 FOR N=0 TO 63
20 PLOT N, 22+20*SIN(N/32*PI)
30 NEXT N
```

THE CHARACTER SET

The letters, digit, punctuation marks, graphic symbols and keyboards are called characters. Most of the characters are single symbols, but there are some which represent a whole word such as the keywords PRINT, PLOT and so on.

There are 256 characters altogether and each one has a code between 0 and 255. To convert between codes and characters, there are two functions, CODE and CHR$.

CODE is applied to a string, and gives the code of the first character in the string (0 if null string).

CHR$ is applied to a number, and gives the character whose code is that number.

This program prints the entire character set.

```
10 FOR I=0 TO 255
20 PRINT CHR$I;
30 NEXT I
```

First at the top you will see the normal characters which you get in normal mode. Then there appear a few keyword and many periods, followed by the same set of characters as at the top but in black on white (inverse video). These are characters you get in graphic mode. Last are the keywords of your computer. Each of the keywords represents a character. You may wonder why there is so many periods in the character set. Actually the series of periods appear on the screen are unused characters or control characters.

This program will print the code of the first character you input

```
10 INPUT A$
20 PRINT CODE A$
30 GOTO 10
```

CHAPTER 6

SLOW AND FAST

Your computer can run at two speeds, SLOW and FAST. When first switched on, after RESET or NEW, the computer runs in the SLOW mode. It can compute and display information on the screen simultaneously.

However you can make it run 4 times faster by allowing it not to display information on the screen. To do this, type

```
FAST
```

Now whenever you press a key, the screen will blink. This is because your computer stops displaying the picture while it works to find out what key you have pressed.

Try this program, you will see the difference between SLOW and FAST.

```
NEW
10 A = INT(16*RND)
20 IF A>=B THEN A = A + 120
30 PRINT CHR$A;
40 GOTO 10
```

Run this program. You will see that the graphic symbols are displayed on the screen one by one until the screen is full and you get the report message "SF in 30".

Now type FAST and run this program again. The screen will blank for about 15 second and the whole picture of random graphic appears suddenly on the screen. Have you count the time this program takes in SLOW mode? If you haven't. Type SLOW to return to SLOW mode and run the program in SLOW mode. Count the time to prove it to yourself that the program takes around one minute.

You may include the SLOW and FAST statement in your program so that your computer may switch between SLOW and FAST mode while running the program. Try the following program.

```
10 SLOW
20 FOR I = 1 TO 64
30 PRINT "A";
40 IF I = 33 THEN FAST
50 NEXT I
60 GOTO 10
```

CHAPTER 7

STRINGS and ARRAYS

STRING ALONG

Would you like to see your name spelled backwards? So far we have played with graphics and numbers, but computers can also manipulate letters and symbols. Your computer can deal with a single character, or it can handle a whole string of characters at a time. This will seem fairly natural, since we humans also usually deal with characters in bunches. Variables which contain character strings, like numeric variables, have names. String variable names follow the same rules as numeric variable names except that they end with a dollar sign ($) and they can have only one character.

Here are some examples of string variable names:

```
A$
B$
```

The variable A is different from the variable A$, and both can be used in the same program.

If you wish the string variable A$ to contain the letters "GOOD MORNING" you can type

```
A$ = "GOOD MORNING"
```

Notice that the characters that you put into a string variable must be enclosed in quotes. The statement

```
PRINT A$
```

will print the contents of the variable A$: in this case, the greeting "GOOD MORNING". Thus, when you have a string of characters that you need often, you can store the string in a variable with a short name.

There are several more computer instructions that manipulate strings. Suppose you want to know the length of a string (how many characters it contains). You can type

```
PRINT LEN "GOOD MORNINGS"
```

or you can type the equivalent statement,

```
PRINT LEN A$
```

and the computer will PRINT the LENgth of the string, in this case 12. Notice that spaces count as characters.

On some occasions you may want to PRINT only a part of A$. To do this you can utilize the substring.

If, for instance, you want to PRINT the first fourletters in A$ you can type

```
PRINT A$(TO 4)
```

and

```
GOOD
```

should appear on the screen.

For each program you write that uses string variables, you must assign the string value within the program. Here's a short program that uses the functions LEN and

```
NEW
10 A$ = "GOOD MORNING"
20 FOR N = 1 TO LEN A$
30 PRINT A$(TO N)
40 NEXT N
```

Type

```
PRINT A$(6 TO)
```

Your computer replies with

```
MORNING
```

Since M is the sixth character in the string and the second number in the bracket is defaulted to be the length of the string.

Try this program

```
NEW
10 A$ = "GOOD MORNING"
20 FOR N = 1 TO LEN A$
30 PRINT A$(N TO)
40 NEXT N
```

Do you get what you expect when the program is RUN?

Suppose you want to PRINT just the "MORN" from the string calles A$.

```
PRINT A$(6 TO 9)
```

The first number (6) specifices the string character space at which the computer is to begin PRINTing. The second number (9) specifices the string character space to stop PRINTing. Thus the instruction is interpreted by the computer as "find the sixth character space in A$, and PRINT characters beginning at the sixth and moving to the right to the ninth".

Consider this program

```
NEW
10 A$ = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
20 PRINT
30 PRINT "TYPE A NUMBER, FROM 1 THROUGH"; LEN A$; " "
40 PRINT "AND I WILL TELL YOU WHICH LETTER HAS THAT
   POSITION IN THE ALPHABET"
50 INPUT P
60 IF P > LEN A$ OR P < 1 THEN GOTO 20
70 PRINT
80 PRINT A$(P); "IS LETTER NUMBER"; P; "IN THE ALPHABET"
```

This program illustrates some common programming practices. Notice the function of the blank spaces in the PRINT statements. What would happen to the output without these blank spaces? Finally, observe that the program limits are always set by LEN A$, rather than the actual number of characters in the alphabet. This allows the program to work even if you specify a different "alphabet" in line 10. Try it and see.

You can substitutes one string for another with a replacement statement such as

```
X$ = A$
```

This statement copies the contents of A$ into X$.

You can also assign substrings, though you cannot do this in some other computers. Type

```
A$ = "ABCDEF"
```

then

```
A$ (2 TO 4) = "1234"
PRINT A$
```

Notice that 123 has replaced BCD in the string A$. The "4" has been left out. This is a characteristic of assigning to substrings: the substring has to be exactly the same length afterwards as it was before. To make sure this happen,· the string that is being assigned to is cut off on the right if it is too long, or filled with spaces if it is too short. Now try

```
A$ (2 TO 4) = "BC"
PRINT A$
```

Notice that there is a space between C and E now.

ADDING STRINGS

It is possible to add a second string to the end of an existing string using the plus (+) sign. Try the following on your computer

```
C$ = "GOOD MORNING"
D$ = C$ + "  " + "JOHN"
PRINT D$
```

Your computer will respond with

```
GOOD MORNING JOHN
```

Adding is especially useful if you wish to take a string apart and then put it back together with slight modifications. For instance, if you wanted to create a new string that was the same as D$ except that the spaces between words would be substituted with dashes, you could type

```
E$ = D$(14 TO 17) + "–" + D$(TO 4) + "–" + D$(6 TO 12)
PRINT E$
```

and

```
JOHN-GOOD-MORNING
```

would appear on your screen.

Try the following program

```
NEW
10 PRINT "TYPE YOUR NAME AND I WILL SHOW IT TO YOU
   SPELLED BACKWARDS
20 INPUT N$
30 R$ = " "
40 REM REVERSE ORDER OF LETTERS
50 FOR T = LEN N$ TO 1 STEP –1
60 R$ = R$ + N$(T)
70 NEXT T
80 PRINT "YOUR NAME SPELLED BACKWARDS IS"; R$
```

MORE STRING FUNCTIONS

Strings can be made up of almost any kind of character, including numbers. However, like items in a PRINT statement, the characters between the quote marks in a string cannot be interpreted arithmetically even if they are numbers. To see what happens when you try, type

```
A$ = "123"
PRINT A$ + 4
```

Your computer confusedly print the error flag and will not execute your statement.

We need the help of the

```
VAL
```

(short for VALue) function to alleviate this program.

The VAL function returns the VALue of the contents of a string as opposed to its actual contents. Type

```
PRINT A$
```

and then type

```
PRINT VAL A$
```

Both commands apparently get the same result; however, appearances can be deceiving. You already know that if you type

```
PRINT A$ + 4
```

Your computer will respond with error flag.

Try typing

```
PRINT VAL A$ + 4
```

and

```
127
```

appears on the screen.

What if you want to put the value of A$ minus 3 into an ordinary (non-string) variable? Simple. Just type

```
D = VAL(A$) – 3
```

Now type

```
PRINT D
```

and see what you get. Are the contents of D as you expect? You can even use VAL to add the numerical value of two different strings. To try this, create a new string

```
B$ = "10"
```

and then type

```
S = VAL A$ + VAL B$
PRINT S
```

Try VAL with different strings, including strings that begin or end with letters.

Sometimes it is necessary to change a number into a string. The STR$ function, which works much like the VAL function in reverse, can be used to make this change. Suppose you want to change the numeric variable P to a string variable. Typing

```
P$ = STR$(P)
PRINT P$
```

will show you how STR$ works. Here is a program that uses STR$ and VAL.

```
290 PRINT "TYPE A NUMBER FROM 1 THROUGH 99999999"
300 INPUT N$
310 N = VAL N$
320 IF N < 1 OR N > 99999999 THEN GOTO 300
330 N$ = STR$ N
340 P$ = " "
350 FOR T = LEN N$ TO 1 STEP -1
360 P$ = P$ + N$(T)
370 NEXT T
380 PRINT
390 PRINT "ORIGINAL", N$
400 PRINT
410 PRINT "REVERSED", P$
420 P = VAL P$
430 PRINT
440 PRINT "ORIGINAL + REVERSED = "; N + P
450 PRINT
460 PRINT
470 GOTO 300
```

INTRODUCTION TO ARRAYS

Arrays enable you to select any element in a table of numbers, and the programming power they give you more than compensates for the bit of thinking and experimenting you must do to become familiar with them.

An array is a table of numbers. The name of this table, called the array name, is any legal variable name: A, for example. The array name A is distinct and separate from the simple variable A. The name of an array must be a single letter.

To create an array, you must first tell the computer the maximum number of elements you want the array to accommodate. To do this you use a DIM statement (DIM stands for DIMension). The elements in an array are numbered from 1, so to DIMension an array called A that will have a maximum of 16 elements, type

```
DIM A(16)
```

The DIM statement above has given us 16 new variables. They behave exactly like the variables you have come to know and love. They are:

```
A(1)
A(2)
A(3)
```

and so on, down to

```
A(16)
```

Although you may find them awkward to type, they can be used just as any other variable. The statement

```
A(9) = 45 + A(12)
```

is perfectly correct. The number in parentheses is called a subscript, and the notation A(12) is read "A-sub-twelve". The subscript can be an arithmetic expression or it can be represented by a variable.

Type the following program. It illustrates the use of variables in the subscript and prints out a display of the contents of each array element.

```
100 REM DIMENSION ARRAY CALLED D TO HOLD 7 NUMBERS
110 DIM D(7)
120 REM FILL THE ARRAY
130 FOR N = 1 TO 7
140 D(N) = N
150 NEXT N
160 REM PRINT THE ARRAY ELEMENTS
170 FOR I = 1 TO 7
180 PRINT "D(";I;") = ";D(I)
190 NEXT I
```

Suppose you want to write a program that generates the numbers from one to eight in scrambled order. To accomplish this you need to manipulate tables of data. This is just the kind of thing for which arrays are excellent. The following program accomplishes this.

```
NEW
200 REM DIMENSION THE ARRAY
210 DIM G(8)
220 REM FILL THE ARRAY
230 FOR I = 1 TO 8
240 G(I) = I
250 NEXT I
260 REM SCRAMBLE THE ARRAY AND CHOOSE EACH ELEMENT
270 FOR W = 1 TO 8
280 REM CHOOSE SOME OTHER ELEMENT
290 MILK = INT (RND * 8) + 1
300 REM WAS MILK DIFFERENT FROM W
310 REM IF NOT, TRY AGAIN
320 IF MILK = W THEN GOTO 290
330 REM INTERCHANGE G(W) AND G(MILK)
340 TEMP = G(W)
350 G(W) = G(MILK)
360 G(MILK) = TEMP
370 NEXT W
380 REM PRINT CONTENTS OF ARRAY
390 FOR C=1 TO 8
400 PRINT G(C)
410 NEXT C
```

Do you understand how this program works? It first fills an array with numbers and then scrambles the contents of the array. Here's a description of what some of the more elusive program lines do. Lines 230 through 250 fill the array and assign each array element a number corresponding to its array number (G(1) = 1, etc.). Line 270 sets the new variable W to numbers 1 through 8. Line 280 sets variable MILK to random integers from 1 to 8. Then line 300 makes sure that the value of W is not equal to the value of MILK at any given time. The contents of variables G(W) and G(MILK) are switched in line 330. Finally the array is printed with lines 390 through 410.

The switching that occurs in line 330 can be thought of like this. Lets say we have two glasses – one is a wine glass (W), and the other is a milk glass (MILK). Oh no, there was a mistake. The milk is in the wine glass and the wine is in the milk glass. Luckily we have an extra glass (TEMP). We can pour the milk into the extra glass, then pour the wine into the wine glass, and finally, pour the milk into the milk glass. Now both drinks have been switched to their proper glasses.

You can have an array of more than one dimension. To set up a two dimensional array B with dimension 2 and 3, you use a DIM statement

```
DIM B(2, 3)
```

This gives you 2 * 3 = 6 subscripted variables

```
B(1, 1)  B(1, 2)  B(1, 3)
B(2, 1)  B(2, 2)  B(2, 3)
```

In fact you can have an array of any dimension. There are also string arrays. The strings in an array differ from simple strings in that they are of fixed length and assignment to them is like assignment to substring, that is, the assignment is cut off from the right if too long or filled with space if too short. The name of a string array is a single character followed by $. A string array and a simple string variable cannot have the same name (unlike the case for numbers).

Suppose you want to have an array A$ of five strings. You have to first decide how many characters you want in each string, suppose 10. Then use the statement

```
DIM A$(5, 10)
```

to set up an array of 5 strings with 10 characters in each string. Type

```
A$(2) = "1234567890"
PRINT A$(2)
```

You get

```
1234567890
```

Now what is the general expression for the substring of a string array element. Type

```
PRINT A$(2) (4 to 8)
```

or

```
PRINT A$(2, 4 TO 8)
```

You get

```
45678
```

Type

```
PRINT A$(2, 7)
```

or

```
PRINT A$(2) (7)
```

You get 7.

ARRAY ERROR MESSAGES

Here are a few error messages you might generate while programming with arrays.

BS (bad subscript error)

If an attempt is made to use an array element that is outside the dimension of the array, this error message will occur. For instance, if A has been dimensioned to 25 with the statement DIM A(25), referring to the element A(52) or any other element whose subscript is less than 0 or greater than 25 will give the BS message.

IR (integer out of range)

You will get this message if you try to use a negative number as an array subscript.

These are some of the ways that you use arrays. The arrays used here are all one dimensional arrays. You can also use arrays that have two or more dimensions.

CHAPTER 8

SAVE and LOAD

When you turn off your computer, you lose all the program and data inside the computer. The only way to save them is to record them on a cassette tape. You can then load the program back on your computer at a later time.

SAVING PROGRAM

Using any ordinary cassette recorder, you can save your program on an ordinary cassette tape. To try it, type in a program, for example, the program to print the character set as shown in chapter 5. Now let's save this program on tape.

1. Connect the MIC socket at the rear of your computer to the MIC socket of your cassette recorder by the pair of cassette cable provided in your computer package. It is better to connect the MIC socket but not the EAR socket because some tape recorders distort the computer's signals if the computer's MIC and EAR leads are connected to the recorder at the same time.

2. Load a blank cassette tape to your recorder. It is advisable to buy high quality tapes for your computer programs, since any imperfections may prevent your program from being reloaded.

3. Type

SAVE "program name" (don't press ENTER now)

You can use any name of any length. Notice that your program name must be within quotation marks.

4. Put your cassette recorder on RECORD and wait for a few seconds to allow the tape to stablize its speed.

5. Now, you press ENTER key to start saving the program. After the ENTER key is pressed, the screen will go blank for about 5 seconds with some thin white lines dancing on the screen. This pattern is followed by horizontal black and white strips jumping around, indicating that your program is being recorded on the cassette tape (Fig 8.1).

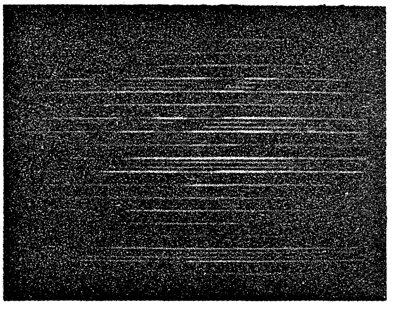


Fig. 8.1

When the recording is over, the screen will go blank and a OK message will appear at the bottom of the screen.

Now you can stop your recorder. Rewind the tape to the beginning. Your program in the computer is not affected in any way by the saving process. You can LIST your program to verify this.

You should check that the program was recorded by listening to the tape. If a lead was not connected, or if you have a bad tape, then you will not get a record. The pattern on the screen indicates only that information is being sent to the recorder. It does not indicate that the information was recorded. If the tape sounds normal, and there is not a lot of background noise, you probably made a good copy.

LOADING PROGRAM

For testing you may reload the program you just saved back to your computer.

1. Connect the EAR sockets on your computer and your recorder.

2. Rewind the tape to the place where you saved your program.

3. Adjust the volumn of your recorder to about ¾ of maximum. If your recorder has base and treble controls, set the treble high and the base low, because computer information is high pitched.

4. Turn on your computer and type a NEW to ensure no program inside the computer's memory. Then type

LOAD "program name" (without ENTER)

You have to type the name the same as the one you saved the program. You can also type

LOAD " " (without ENTER)

This command load the next program on tape, regardless of its name.

5. Press the PLAY key on your recorder and the ENTER key on your computer to start loading.

During loading, you see many thin black lines slanting across the screen as shown in Fig. 8.2. This pattern indicates that the computer is waiting for the information. After a few seconds, a different pattern (Fig. 8.3) which looks like thick black bars dancing up and down with black lines. This indicates that the computer is reading data from tape. If you cannot get this pattern, your recorder volumn may be too high or too low. Adjust until black strips appear.

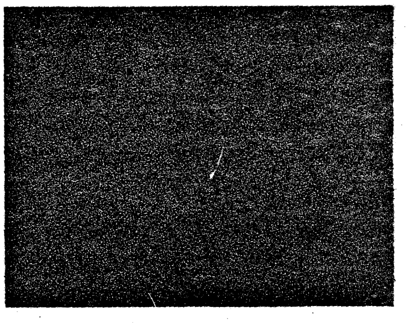


Fig. 8.2

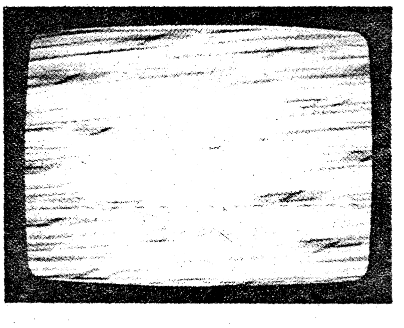


Fig. 8.3

After about 15 seconds, the bars will disappear and a OK message will appear at the bottom of the screen indicating that the program has been successfully loaded. To ensure your program is loaded, type LIST to list your program.

MEMORY REQUIREMENTS

Your computer has a standard 2K bytes of memory built-in. It is more than enough to run a program as short as the program examples in this book. But some programs need more than 2K bytes of memory to run it. If you run a program more than 2K bytes but less than 16K bytes, you have to buy a 16K RAMPACK.

It is important to point out that your computer can run all the BASIC programs prerecorded tape for SINCLAIR/TIMEX 1000 microcomputer. Most of these tapes require 16K bytes of memory and are commonly advertised in magazines like Syntax ans Sync. More tapes of games, business and education will soon be available.

However, due to different usage of memory and coding of keywords, programs saved by your computer cannot be read by SINCLAIR/TIMEX 1000, but BASIC programs saved by SINCLAIR/TIMEX 1000 can be loaded into your computer and run properly.

CHAPTER 9

The computer and Music

One of it's excellent features of your computer is it's programmable sound feature.

Up to now, you have heard so many different sound from your computer. When you turn it on, it sounds a "beep!"; each time you press a key, you get a sound feedback of difference frequency from different key. When you type in a error command/statement, it gives a buzzer sound. But the most important thing is that you can control the computer's sound output by your program, even you can ask it to perform any melody!

BEEP Mode and NO BEEP Mode

"So much noise!" you may think, "can I play my computer more quitely?" The answer is positive. You can simply type

NOBEEP

then ENTER. Now the computer is in NO BEEP mode, and no sound will be responded when you press the keys. Later, if you like to have beep sound again, you type

BEEP

then ENTER. Now the computer is in BEEP mode, and the computer will respond with it's lively sound again if any key is activated.

Turn ON the computer or a RESET cycle will automatically set the computer into BEEP mode.

Using MUSIC Statement

Most of the modern computer magazines are recently talking about some interesting title; Computer & Music, Computer & Poem, Computer & Literature, etc. The excellent design of your computer will bring you to enter one of these interesting area: Computer & Music.

The MUSIC statement in your computer's BASIC will help you to perform some music tones, even a melody, via the speaker inside the computer. To try it, you enter MUSIC followed by some strings specified. For example

MUSIC "C10D10E10F10G10"

perform a "1-2-3-4-5:: (Pronounce as do-re-mi-fa-so) of 10 units of time. The string (remember that a string must be inside the quotation) represents a series of notes, each can be 2 to 4 characters long. The first one or two characters represent the note and the octave, sharps are represented by the inverse video characters. Following the note are one or two digits specifying how many units of time the note is to last, and can be any value form 1 to 99 (a 0 is equivalent to 256). All the possible notes and their frequencies are listed in the Appendix A.

A rest (pause) is just like an ordinary note but with a "." replacing the note characters. Spaces are ignored in interpreting the string and empty string is allowed.

If you have entered an incorrect string format for MUSIC statement, you will get a MF error report on the screen.

TEMPO in your MUSIC

The TEMPO statement is used to control the note duration unit used in the MUSIC statement. For example, if you type in

TEMPO 20

you have got the note duration unit equal to 20x3.94 ms. Normally the TEMPO format is as

```
TEMPO n
```

where 0< n< 255 and n=0 is equivalent to n=256. If you have typed a number outside this limit you will get an IR error.

For example type and run this program:

```
NEW
10 MUSIC "C10D10E10F10G10"
20 RUN
```

Your short melody will go forward without stop. It is because the RUN statement in line 20 cause a loop between line 10 and line 20, and it is an infinite loop. To stop this program you must use the BREAK command, it is a shift key up-on the SPACE key on the keyboard. (You can also use the RESET key to stop the program, but it's not a good idea to break a program by RESET, because in some circumstances, RESET can clear some data in your program).

Now, BREAK it then type (followed by an ENTER)

```
TEMPO 10
```

and run the short program again to see what happens. As you can hear, the music goes faster. You can BREAK It again and type TEMPO 100, or TEMPO 40, TEMPO 5 to see how the tempo statement effect. Try it yourself.

You can also use the TEMPO statement inside your program to perform a more animating music.

Programming your own melody.

Use the character – Note table listed in Appendix A, you can easily program out any melody you like and use the TEMPO statement to perform it lively. For example, type

```
NEW
10 REM ... OLD SUSANNA
20 LET A$ = "C6D2E3G3G6A2G3E3C6D2E3E3D3C3D12"
30 LET N = 1
35 TEMPO 30
40 MUSIC A$
45 LET N = N + 1
50 IF N < 3 THEN GOTO 40
60 IF N > 3 AND N < 6 THEN GOSUB 200
70 IF N = 6 THEN GOSUB 250
80 IF N < 6 THEN GOTO 40
100 STOP
200 TEMPO 10
210 MUSIC A$
220 RETURN
250 FOR I = 1 TO 3
260 TEMPO 25 / I
270 MUSIC A$
280 NEXT I
290 RETURN
```

SOUND statement

The SOUND statement can use for special sound effect for your games and even in your music. The normal format of SOUND statement is

SOUND m, n

where m, n is an integer

$0 < m < 255$
$0 < n < 65535$

(m=0 is equivalent to m=256; and n=0 is equivalent to n=65536). When m, n is specified, the sound statement produces a sound of frequency equal to 32500/m Hz and its duration equal to n/65 ms in the FAST mode. In computer and display mode (called SLOW mode), the duration is approximately four times longer and the sound is modulated by 50 Hz. For example, type this program and run it.

```
FAST
10 FOR M = 100 TO 200 STEP 10
20 FOR N = 100 TO 1000 STEP 10
30 SOUND M, N
40 NEXT N
50 NEXT M
60 STOP
```

Note that there are two loops inside this program; M loop and N loop. N loop occurs entirely within the M loop, that is what we called Nested Loops. Nested loops are very useful in your programming.

CHAPTER 10

PEEK AND POKE

Your computer consists of five chips all together, a Z80 processor, a ROM, 2K RAM, a 74LS05 TTL and a computer controllor chip. The Z80 processor is the brain of your computer. It does all the arithematic, decides what key you have pressed and refreshes the TV. It does this by fetching and executing instructions from the ROM. The instructions are stored in the form of long sequence of bytes (a byte is a number between 0 and 255). Each byte has an address where you can find the byte. The address of the first byte is 0 and the address of the last one is 8192. That is why the basic of your computer called a 8K BASIC. (8K = 8*1024)

One other chip is the RAM. It is where the computer stores the information it wants to keep. The information are also stored in the form of bytes. Each byte has an address between 16384 and 49151 (fully expanded to 32K RAM). For your basic system, you have got 2K RAM only, address from 16384 to 18431. To expand to 32K RAM, you have to add a RAM PACK to your computer.

Sometimes you may like to know the value of the byte stored in a given address. This can be done by using the function PEEK. For example this program prints out the address and the contents of the first 20 bytes in the ROM.

```
10 PRINT "ADDRESS"; TAB8; "BYTE"
20 FOR A = 0 TO 19
30 PRINT A; TAB8; PEEK A
40 NEXT A
```

Sometimes you may like to write a byte into the RAM (Note that you cannot write a byte into the ROM). This can be done by the instruction POKE. Type

```
POKE 17300,50
```

This makes the byte at 17300 has the value 50. You can confirm the byte is really 50 by typing.

```
PRINT PEEK 17300
```

USR

This chapter is written for those that understand Z80 machine code, the set of instructions that Z80 processor chip uses.

Machine code routines can be executed from within a basic program using the function USR n where n is a number between 0 and 65535. It is the starting address of the rountine, and its result is a two byte unsigned integer, the contents of the bc register pair on return. The return address to the BASIC is stacked in the usual way, so return is by a Z80 RET instruction.

There are certain restrictions on USR routines:

A USR routine should not use the a', f', ix, iy & r registers which the display routine uses.

Your machine language program should be located at a proper place of the memory so as not to be overwritten by the BASIC system. Some safe place are as follow:

(i) In a REM statement:

type in a REM statement with enough characters to hole your machine code, which you then poke in. Make this first line in the program, or it might move about. Avoid halt instructions, since this will be recognized as the end of the REM statement.

(ii) In a string:

set up a long enough string and then assign a machine code byte to each character. String will move about in the memory.

(iii) At the top of the memory:

when your computer is first switched on, it tests how much memory there is and stores the first non-existant byte in the location 16388 and 16389, which is known as RAMTOP. On the other hand NEW just checks the memory up to the byte before the address in RAMTOP. So you can poke the address of an existent byte into RAMTOP, and after New the bytes beyond that address in RAMTOP is left out of Basic. You can safely put your machine language in these area.

To see the address in the RAMTOP byte, type

```
PRINT PEEK 16388+256*PEEK16389
```

For a basic 2K system, you should get 18432, the address of the first non existent byte.

Now suppose you have a machine language program of 18 bytes. Poke the value 71 into 16389 and type NEW. That will leave you 256 bytes outside BASIC.

Try this program to see what does it do?

```
POKE 16389,71
NEW
 10 POKE 18180,33
 20 POKE 18181,213
 30 POKE 18182,65
 40 POKE 18183,54
 50 POKE 18184,45
 60 POKE 18185,35
 70 POKE 18186,54
 80 POKE 18187,42
 90 POKE 18188,35
100 POKE 18189,54
110 POKE 18190,49
120 POKE 18191,35
130 POKE 18192,54
140 POKE 18193,49
150 POKE 18194,35
160 POKE 18195,54
170 POKE 18196,52
180 POKE 18197,201
190 X=USR18180
```

A greeting message will appear at the middle of the screen.

Here is an example to locate the same machine language program in a REM statement. Type

```
NEW
10 REM 5AAQH7QE7QL7QL7QOA
20 POKE 17308,213
30 POKE 17309,65
40 POKE 17324,201
50 X=USR17307
```

Run this program and you see that the greeting message "HELLO" appears in the middle of the screen again. Now press ENTER to see the listing of your program, you may discover that some letters in your REM statement has changed to character strings. Do you know why?

Now type in the following program without NEW.

```
100 FOR I = 17307 TO 17324
110 PRINT PEEK I
120 NEXT I
```

Type RUN 100. The machine code of your program starting at 17307 is displayed.

APPENDIX A

BASIC Functions and Statements

Function	Type of argument (X)	Operation
ABS	number	Returns absolute value .
ACS	number	Returns arc cosine in radians. AG error if X not in the range -1 to $+1$.
ASN	number	Returns arc sine in radians. AG error if X not in the range -1 to $+1$.
ATN	number	Returns arc tangent in radians.
CHR$	number	Returns the character whose code is X, rounded to the nearest integer. IR error if X not in the range 0 to 255.
CODE	string	Returns the code of the first character in X (or 0 if X is the empty string).
COS	number (in radians)	Returns the cosine value of X .
EXP	number	Returns the value of e^x.
INKEY$	none	Reads the keyboard and returns the character representing (in normal mode) the key pressed if there is exactly one, otherwise the null string.
INT	number	Returns the integer part (always rounds down).
LEN	string	Returns length of the string (0 if null string).
LOG	number	Returns natural logarithm (base e) . AG error if $X \leqslant 0$.
PEEK	number	Returns the value of the byte in memory whose address is X (round to the nearest integer). IR error if X not in the range 0 to 65535.
PI	none	Returns the value of π (3.14159265)
RND	none	Returns the next pseudo-random number Y in a sequence generated by taking the powers of 75 modulo 65537, subtracting 1 and dividing by 65536. $0 \leqslant Y < 1$.
SGN	number	Returns the sign (-1, 0 or $+1$) of X.
SIN	number (in radians)	Returns the sine value of X.
SQR	number	Returns square root of X . AG error if $X < 0$.
STR$	number	The string of characters that would be displayed if X were printed.
TAN	number (in radians)	Returns the tangent value of X .
USR	number	Branch to machine code subroutine whose starting address is X (rounded to the nearest integer). On return, the result is the contents of the BC register pair. IR error if X is not in the range 0 to 65535.
VAL	string	Evaluates X (without its bounding quotes) as a numerical expression. IE error if X contains a syntax error, or gives a string value. Other errors possible, depending on the expression.

STATEMENTS

	in this list
a	represents a single letter
v	represents a variable
e	represents an expression
f	represents a string valued expression
x,y,z	represents numerical expressions
m,n	represents numerical expressions that are rounded to the nearest integer
s	represents a statement

Note that arbitrary expression are allowed everywhere (except for the line number at the beginning of a statement).

BEEP	Enables the keyboard audio feedback. A beep of different frequencies will be produced for different keys pressed while entering program lines, commands or INPUT data.
CLEAR	Set numeric variables to zero, strings to null.
CLS	(Clear Screen) Clears the display file.
CONT	Suppose "XX IN YY" was the last report. Then CONT has the effect GOTO YY if XX $\neq$ ST GOTO YY+1 if XX = ST (STOP statement)
COPY	If the printer is attached, a copy of the video display is printed; otherwise does nothing. COPY command does not clear the screen first. There must be no spaces before COPY. Report BK if BREAK pressed.
DIM a $(n_1, ..., n_k)$	Allocate storage for an array a of numbers with k dimensions, n1,..., nk. Initializes all the values to 0. OM error occurs if there is no space to fit the array in. An array is undefined until it is dimensioned in a DIM statement.
DIM a$ $(n_1, ..., n_k)$	Allocate storage for an array a $ of characters with k dimensions, n1,..., nk. Initializes all the values to " ". This can be considered as an array of strings of fixed length n_k, with k-1 dimensions $n_1, ..., n_{k-1}$. OM error occurs if there is no room to fit the array in. An array is undefined until it is dimensioned in a DIM statement.
FAST	Put the computer in fast mode. The display file is displayed only at the end of the program, while INPUT data is being typed in, or during a pause.
FOR a=x TO y	FOR a=x to y STEP 1
FOR a=x TO y STEP z	Deletes any simple variable a, and sets up a control variable with value x, limit y, step z, and looping address 1 more than the line number of the FOR statement (−1 if it is a command). See NEXT a. OM error occurs if there is no room for the control variable.

GOSUB n	Branch to subroutine beginning at n. OM error can occur if there are not enough RETURNs.
GOTO n	Branch to n (or, if there is none, the first line after that)
IF x THEN s	If x is true (non-zero) then s is executed. The form 'IF x THEN line number' is not permitted
INPUT v	Stops (with no special prompt) and await input from keyboard; the value of this is assigned to v. In fast mode, the display file is displayed. INPUT cannot be used as a command; II error occurs if you try. If BREAK is pressed, the program stops with report BK. Assigns the value of e to the variable v. "LET" is optional. A variable has to be assigned in a LET or INPUT statement, else it will be undefined. If v is a subscripted string variable, or a sliced string variable (substring), then the assignment is Procrustean: the string value of e is either truncated or filled out with spaces on the right, to make it the same length as the variable v.
LIST	LIST 0
LIST n	Starting at line n, lists the program to the screen, and makes n the current line. OM or SF error if the listing is too long to fit on the screen; CONT will do exactly the same again.
LLIST	LLIST 0
LLIST n	Like LIST, but using the printer instead of the screen. Nothing should do if the printer is not attached. Stops with report BK if BREAK pressed.
LOAD f	Search a program called f on tape, and loads it and its variables. If f=" ", then loads the first program available. If BREAK is pressed or a tape error is detected, then (i) if no program has yet been read in from tape, stops with report BK and old program; (ii) if part of a program has been read in, then executes NEW.
LPRINT ...	Like PRINT, but using the printer instead of the screen. A line of text is sent to the printer. (i) when printing spills over from one line to the next, (ii) after an LPRINT statement that does not end in a comma or a semicolon, (iii) when TAB item or a comma requires a new line, or (iv) at the end of the program, if there is anything left unprinted. In an AT item, the line number is ignored and only the column number has any effect. An AT item never sends a line of text to the printer.
MUSIC f	Produces a melody from the speaker according to the content of the string f and the preselected tempo. The string represents a series of

notes, each 2 to 4 characters long. The first one or two characters represent the note and the octave. Sharps are represented by the inverse video characters. The following table gives all the possible notes and their frequencies.

Characters	Note	Frequency (Hz)
C <	$\underset{\cdot}{1}$	131.Ø
$\boxed{C}$ <	# $\underset{\cdot}{1}$	138.3
D <	$\underset{\cdot}{2}$	147.1
$\boxed{D}$ <	# $\underset{\cdot}{2}$	155.5
E <	$\underset{\cdot}{3}$	165.Ø
$\boxed{E}$ <	$\underset{\cdot}{4}$	174.7
F <	$\underset{\cdot}{4}$	174.7
$\boxed{F}$ <	# $\underset{\cdot}{4}$	184.7
G <	$\underset{\cdot}{5}$	195.8
$\boxed{G}$ <	# $\underset{\cdot}{5}$	2Ø7.Ø
A <	$\underset{\cdot}{6}$	219.6
$\boxed{A}$ <	# $\underset{\cdot}{6}$	233.8
B <	$\underset{\cdot}{7}$	246.2
$\boxed{B}$ <	1	262.1
C	1	262.1
$\boxed{C}$	# 1	277.8
D	2	295.5
$\boxed{D}$	# 2	312.5
E	3	331.6
$\boxed{E}$	4	349.5
F	4	349.5
$\boxed{F}$	# 4	369.3
G	5	391.6
$\boxed{G}$	# 5	416.7
A	6	439.2
$\boxed{A}$	# 6	471.Ø
B	7	492.4
$\boxed{B}$	$\dot{1}$	524.2
C >	$\dot{1}$	524.2
$\boxed{C}$ >	# $\dot{1}$	56Ø.3
D >	$\dot{2}$	59Ø.9
$\boxed{D}$ >	# $\dot{2}$	625.Ø
E >	$\dot{3}$	663.3
$\boxed{E}$ >	$\dot{4}$	7Ø6.5
F >	$\dot{4}$	7Ø6.5
$\boxed{F}$ >	# $\dot{4}$	738.6
G >	$\dot{5}$	792.7
$\boxed{G}$ >	# $\dot{5}$	833.3
A >	$\dot{6}$	878.4

$\boxed{A}$	>	# $\dot{6}$	955.9
B	>	$\dot{7}$	984.8
$\boxed{B}$	>	$\dot{\dot{1}}$	1∅48.4

Following the note are one or two digits specifying how many units of time the note is to last. This note duration unit can be changed by the TEMPO statement, and is initialized to approximately 98.5mS after reset (TEMPO 25). The one or two digits can be from 1 to 99 (a 0 is equivalent to 256).

A rest (pause) is just like an ordinary note but with a "." replacing the note characters. Spaces are ignored in interpreting the string and empty string is allowed.

MF error results if the format of the string is incorrect.

NEW — Delete entire program and reset all variables, using the memory up to but not including the byte whose address is in the system variable RAMTOP (bytes 16388 and 16389).

NEXT a
(i) Finds the control variable a.
(ii) Adds its step to its value.
(iii) If the step $\geqslant$ 0 and the value $\leqslant$ the limit; or if the step $<$ 0 and the value $\geqslant$ the limit, then jumps to the looping line.

NF error if there is a simple variable a.
UV error if there is no simple or control variable a.

NOBEEP — Turns off the keyboard audio feedback feature. No beep will be produced on pressing the keys.

PAUSE n — Stops computing for n/50 seconds (or n/60 seconds for USA system) or until a key is pressed.

$0 \leqslant n \leqslant 65535$, else IR error, If $n \geqslant 32768$ then the pause is not timed, but lasts until a key is pressed.

PLOT m,n — Turns on the pixel (m,n); moves the PRINT position to just after that pixel. (0,0) is lower left hand corner.

$0 \leqslant m \leqslant 63$, $0 \leqslant n \leqslant 43$ else IR error

POKE m,n — Loads the value n to the byte in memory with address m.

$0 \leqslant m \leqslant 65535$, $0 \leqslant n \leqslant 255$ else IR error.

PRINT ... — The '...' is a sequence of PRINT items which are written to the display file for display on the television. The items are seperated by commas or semicolons. The position (line and column) where the next character is to be printed is called the PRINT position.

A PRINT item can be
(i) empty, i.e. nothing
(ii) a numerical expression

First, if the value is negative, a minus size is printed. Now let X be the modulus of the value.

If $x < 10^{-5}$ or $x \geqslant 10^{13}$, then it is printed using expontential notation. The mantissa part has up to eight digits (with no trailing zeros), and the decimal point (absent if only one digit) is after the first. The exponent part is E, followed by + or –, followed by one or two digits.

Otherwise x is printed in ordinary decimal notation with up to eight significant digits, and there are no trailing zeros after the decimal point. A decimal point right at the beginning is always followed by a zero, so for instance 0.02 and 0.2 are printed as such.

0 is printed as a single digit 0.

(iii) a string expression.
The tokens in the string are expanded, possibly with a space before or after.

Unused characters and control characters are printed as ".".

(iv) AT m,n
The PRINT position is changed to line m (mounting from the top), column n (counting from the left).

$0 \leqslant m \leqslant 21$, else SF error if m = 22 or 23, IR error otherwise.

$0 \leqslant n \leqslant 31$, else IR error.

(v) TAB n
n is reduced with modulo 32. The PRINT position is then moved to column n, staying on the same line unless this would involve backspacing, in which case it moves on to the next line.
$0 \leqslant n \leqslant 255$, else IR error. The PRINT position remains unchanged if there is a semicolon between two items, so that the second item follows on immediately after the first. A comma, on the other hand, moves the PRINT position on at least one place, and after that, however many as are necessary to leave it in column 0 to 16, throwing a new line if necessary.

If a PRINT statement does not end in a semicolon or comma, a new line is thrown.

OM error may occur with 1½K or less of memory.

SF error means that the screen is filled. In both cases, the cure is CONT, which will clear the screen and carry on.

RND — Yields a random number X, $0 \leqslant X < 1$.

RAND n — Assigns a value n to the system variable (called SEED) used to generate the next value of RND if $n \neq 0$. If $n = 0$ then it is given the value of another system variable (called FRAMES) that counts the frames so for displayed on the television, and so should be fairly random.

IR error occurs if n is not in the range 0 to 65535.

REM ... — Remark statement '...' can be any sequence of characters except 'ENTER'.

RETURN — Pops a line number from the GOSUB stack, and branches to the after it.

RG error occurs when there is no line number on the stack.

RUN — RUN 0.

RUN n — CLEAR, and then GOTO n.

SAVE f — Saves the program and variables on tape, and calls it f. SAVE should not be used inside a GOSUB routine.

If f is the empty string, NA error occurs.

SCROLL — Scrolls the display file up one line, losing the top line and making an empty line at the bottom. PRINT position is now at the start of that empty line.

SLOW	Puts the computer into compute and display mode. The display file is displayed continuously, and computing is done during the blank lines at the top and bottom of the picture.
SOUND m,n	Produces a sound of frequency equal to 32500/m Hz and duration equal to n/65mS in the fast mode. In compute and display mode, the duration is approximately four times longer and the sound is modulated by 50 Hz. $0 \leqslant m \leqslant 255$, $0 \leqslant n \leqslant 65535$, else IR error. m=0 is equivalent to m=256 and n=0 is equivalent to n=65536.
STOP	Stops the program with an ST report. CONT will resume with the following line.
TEMPO n	Sets the note duration unit used in the MUSIC statement to n x 3.94mS. After reset a "TEMPO 25" statement is executed automatically. $0 \leqslant n \leqslant 255$, else IR error. n=0 is equivalent to n=256.
UNPLOT m,n	The pixel (m, n) will be blanked out instead of being turned on.

APPENDIX B

REPORT MESSAGES

After a command is executed or after a program execution is completed or interrupted, a report message will be displayed showing what has happened and where in the program it happened. If a command is executed then only the message without the line number is displayed. The message remains until a key is pressed.

The following table gives each report message with a general description and a list of the situations where it can occur.

Message	Meaning	Situation
OK	Successful completion of a program, or jump to line number bigger than any existing. A report with "OK" does not change the line number used by CONT.	Any
NF	NEXT without FOR. NEXT is used without a matching FOR statement.	NEXT
UV	Undefined variable. This will happen when (i) a simple variable is used before it has been assigned in a LET statement; (ii) a subscripted variable is used before it has been assigned in a DIM statement; or (iii) a control variable is used before it has been set up as a control variable in a FOR statement, while there is no ordinary simple variable with the same name.	Any
BS	Bad subscript. Attempt to assign a matrix element with a subscript a beyond the dimensioned range.	Subscripted variables
OM	Out of memory. Due to shortage of memory, the report message may be incomplete on the screen.	LET, INPUT, DIM, PRINT, LIST, PLOT, UNPLOT, FOR, GOSUB. Sometimes during function evaluation.
SF	Screen full. CONT will make room by clearing the screen.	PRINT, LIST
OV	Overflow. The number calculated is greater than about 10^{38}.	Any arithmetic
RG	RETURN without GOSUB. A RETURN statement is encountered before a matching GOSUB is executed.	RETURN
II	Illegal INPUT. Attempt to use INPUT as a direct command.	INPUT
ST	STOP statement executed. CONT will execute the next statement.	STOP
AG	Invalid argument to certain functions.	SQR, LOG, ASN, ACS, **

IR	Integer out of range. When an integer is required, the floating point argument is rounded to the nearest integer. If IR error results if this is outside a suitable range.	RUN, RAND, POKE, DIM, GOTO, GOSUB, LIST, LLIST, PAUSE, PLOT, UNPLOT, CHR$, PEEK, USR, SOUND, TEMPO
IE	Invalid expression The text of the (string) argument of VAL does not form a valid numerical expression.	VAL
BK	Program interrupted by "BREAK" key. In certain circumstances, the "SPACE" key acts as "BREAK". This is recognized: (a) at the end of a statement while a program is running. (b) while the computer is looking for a program on tape, or (c) while the computer is using the printer.	At end of any statement, or in LOAD, SAVE, LPRINT, LLIST, COPY or INPUT
NA	Program name provided is the null string which is not allowed.	SAVE
MF	Music string format incorrect.	MUSIC

APPENDIX C

Order of Priority of Operators

OPERATORS

The following are operators and their priorities:

Operator	Operation	Priority
**	Raising to a power AG error if the left operand is negative	10
—	Unary minus	9
*	Multiplication	8
/	Division	8
+	Addition (on numbers), or concatensation (on strings)	6
—	Subtraction	6
=	Equal to	5
>	Greater than	5
<	Less than	5
< =	Less than or equal to	5
> =	Greater than or equal to	5
< >	Not equal to	5
	(Both operands must be of the same type for all the relational operators. The result is a number, 1 if the comparison holds, else 0.)	
NOT	Operand always a number. Gives 1 if operand 0, else 0	4
AND	Right operand always a number. Numeric left operand: A AND B = A if B $\neq$ 0 0 if B = 0 String left operand: A$ AND B = A$ if B $\neq$ 0 " " if B = 0	3
OR	Both operands numbers A OR B = 1 if B $\neq$ 0 A if B = 0	3

APPENDIX D

The character set

This is the complete character set, with codes in decimal & hex. If one imagines the codes as being Z80 machine code instructions, then the right hand columns give the corresponding assembly language mnemonics. As you are probably aware if you understand these things, certain Z80 instructions are compounds starting with CBh or EDh; the two right hand columns give these.

Code	Character	Hex	Z80 assembler	– after CBh
0	space	00	nop	rlc b
1		01	ld bc, NN	rlc c
2		02	ld (bc), a	rlc d
3		03	inc bc	rlc e
4		04	inc b	rlc h
5		05	dec b	rlc l
6		06	ld b,N	rlc (hl)
7		07	rlca	rlc a
8		08	ex af, af'	rrc b
9		09	add hl, bc	rrc c
10		0A	ld a,(bc)	rrc d
11	"	0B	dec bc	rrc e
12		0C	inc c	rrc h
13	$	0D	dec c	rrc l
14		0E	ld, c,N	rrc (hl)
15		0F	rrca	rrc a
16	(	10	djnz DIS	rl b
17	)	11	ld de,NN	rl c
18	>	12	ld (de), a	rl d
19	<	13	inc de	rl e
20	=	14	inc d	rl h
21	+	15	dec d	rl l
22	–	16	ld d,N	rl (hl)
23	*	17	rla	rl a
24	/	18	jr DIS	rr b
25	;	19	add hl, de	rr c
26	,	1A	ld a, (de)	rr d
27	.	1B	dec de	rr e
28	0	1C	inc e	rr h
29	1	1D	dec e	rr l
30	2	1E	ld e,N	rr (hl)
31	3	1F	rra	rr a
32	4	20	jr nz, DIS	sla b
33	5	21	ld hl,NN	sla c
34	6	22	ld (NN), hl	sla d
35	7	23	inchl	sla e
36	8	24	inch	sla h
37	9	25	dech	sla l
38	A	26	ld, h,N	sla (hl)
39	B	27	daa	sla a

Code	Character	Hex	Z80 assembler	– after CBh	– after EDh
40	C	28	jr z,DIS	sra b	
41	D	29	add hl,hl	sra c	
42	E	2A	ld hl,(NN)	sra d	
43	F	2B	dec hl	sra e	
44	G	2C	inc l	sra h	
45	H	2D	dec l	sra l	
46	I	2E	ld,l,N	sra (hl)	
47	J	2F	cpl	sra a	
48	K	30	jr nc,DIS		
49	L	31	ld sp,NN		
50	M	32	ld (NN),a		
51	N	33	inc sp		
52	O	34	inc (hl)		
53	P	35	dec (hl)		
54	Q	36	ld (hl), N		
55	R	37	scf		
56	S	38	jr c,DIS	srl b	
57	T	39	add hl,sp	srl c	
58	U	3A	ld a,(NN)	srl d	
59	V	3B	dec sp	srl e	
60	W	3C	inc a	srl h	
61	X	3D	dec a	srl l	
62	Y	3E	ld a,N	srl (hl)	
63	Z	3F	ccf	srl a	
64	THEN	40	ld b,b	bit 0,b	in b,(c)
65	TO	41	id b,c	bit 0,c	out (c), b
66	STEP	42	ld b,d	bit 0,d	sbc hl,bc
67	RND	43	ld b,e	bit 0,e	ld (NN),bc
68	INKEY$	44	id b,h	bit 0,h	neg
69	PI	45	ld b,l	bit 0,1	retn
70		46	ld b,(hl)	bit 0,(hl)	im 0
71		47	ld b,a	bit 0,a	ldi, a
72		48	ld c,b	bit 1,b	in c,(c)
73		49	ld c,c	bit 1,c	out (c),c
74		4A	ld c,d	bit 1,d	adc hl,bc
75		4B	ld c,e	bit 1,e	ld bc,(NN)
76		4C	ld c,h	bit 1,h	
77		4D	ld c,l	bit 1,l	reti
78		4E	ld c,(hl)	bit 1,(hl)	
79		4F	ld c,a	bit 1,a	ld r, a
80		50	ld d,b	bit 2,b	in d,(c)
81		51	ld d,c	bit 2,c	out (c),d
82	not used	52	ld d,d	bit 2,d	sbc hl,de
83		53	ld d,e	bit 2,e	ld (NN),de
84		54	ld d,h	bit 2,h	
85		55	ld d,l	bit 2,l	
86		56	ld d,(hl)	bit 2,(hl)	im 1
87		57	ld d,a	bit 2,a	id a,i
88		58	ld e,b	bit 3,b	in e,(c)
89		59	ld e,c	bit 3,c	out (c),e
90		5A	ld e,d	bit 3,d	adc hl,de
91		5B	ld e,e	bit 3,e	ld de,(NN)
92		5C	ld e,h	bit 3,h	
93		5D	ld e,l	bit 3,l	
94		5E	ld e,(hl)	bit 3,(hl)	im 2

Code	Character	Hex	Z80 assembler	– after CBh	– after EDh
95		5F	ld e,a	bit 3,a	id a,r
96		60	ld h,b	bit 4,b	in h,(c)
97		61	ld h,c	bit 4,c	out (c),h
98		62	ld h,d	bit 4,d	sbc hl,hl
99		63	ld h,e	bit 4,e	ld (NN),hl
100		64	ld h,h	bit 4,h	
101		65	ld h,l	bit 4,l	
102		66	ld h,(hl)	bit 4,(hl)	
103	not used	67	ld h,a	bit 4,a	rrd
104		68	ld l,b	bit 5,b	in l,(c)
105		69	ld l,c	bit 5,c	out (c),l
106		6A	ld l,d	bit 5,d	adc hl,hl
107		6B	ld l,e	bit 5,e	ld de,(NN)
108		6C	ld l,h	bit 5,h	
109		6D	ld l,l	bit 5,l	
110		6E	ld l,(hl)	bit 5,(hl)	
111		6F	ld l,a	bit 5,a	rld
112	cursor up ↑	70	ld (hl),b	bit 6,b	
113	cursor down ↓	71	ld (hl),c	bit 6,c	
114	cursor left ←	72	ld (hl),d	bit 6,d	sbc hl,sp
115	cursor right →	73	ld (hl),e	bit 6,e	ld (NN),sp
116	GRAPHICS	74	ld (hl),h	bit 6,h	
117	EDIT	75	ld (hl),l	bit 6,l	
118	ENTER	76	halt	bit 6,(hl)	
119	DELETE	77	ld (hl),a	bit 6,a	
120	LMODE	78	ld a,b	bit 7,b	in a(c)
121	BREAK	79	ld a,c	bit 7,c	out (c),a
122	LINE NO.	7A	ld a,d	bit 7,d	adc, hl,sp
123	not used	7B	ld a,e	bit 7,e	ld sp,(NN)
124	not used	7C	ld a,h	bit 7,h	
125	not used	7D	ld a,l	bit 7,l	
126	number	7E	ld a,(hl)	bit 7,(hl)	
127	cursor	7F	ld a,a	bit 7,a	
128		80	add a,b	res 0,b	
129		81	add a,c	res 0,c	
130		82	add a,d	res 0,d	
131		83	add a,e	res 0,e	
132		84	add a,h	res 0,h	
133		85	add a,l	res 0,l	
134		86	add a,(hl)	res 0,(hl)	
135		87	add a,a	res 0,a	
136	inverse	88	adc a,b	res 1,b	
137		89	adc a,c	res 1,c	
138		8A	adc a,d	res 1,d	
139	inverse "	8B	adc a,e	res 1,e	
140	inverse	8C	adc a,h	res 1,h	
141	inverse $	8D	adc a,l	res 1,l	
142	inverse	8E	adc a,(hl)	res 1,(hl)	
143	inverse	8F	adc, a,a	res 1,a	
144	inverse (	90	sub b	res 2,b	
145	inverse)	91	sub c	res 2,c	
146	inverse >	92	sub d	res 2,d	
147	inverse <	93	sub e	res 2,e	
148	inverse =	94	sub h	res 2,h	
149	inverse +	95	sub l	res 2,l	

Code	Character	Hex	Z80 assembler	— after CBh	— after EDh
150	inverse –	96	sub (hl)	res 2,(hl)	
151	inverse *	97	sub a	res 2,a	
152	inverse /	98	sbc a,b	res 3,b	
153	inverse ;	99	sbc a,c	res 3,c	
154	inverse ,	9A	sbc a,d	res 3,d	
155	inverse .	9B	sbc a,e	res 3,e	
156	inverse 0	9C	sbc a,h	res 3,h	
157	inverse 1	9D	sbc a,l	res 3,l	
158	inverse 2	9E	sbc a,(hl)	res 3,(hl)	
159	inverse 3	9F	sbc a,a	res 3,a	
160	inverse 4	A0	and b	res 4,b	
161	inverse 5	A1	and c	res 4,c	cpi
162	inverse 6	A2	and d	res 4,d	ini
163	inverse 7	A3	and e	res 4,e	outi
164	inverse 8	A4	and h	res 4,h	
165	inverse 9	A5	and l	res 4,l	
166	inverse A	A6	and (hl)	res 4,(hl)	
167	inverse B	A7	and a	res 4,a	
168	inverse C	A8	xor b	res 5,b	ldd
169	inverse D	A9	xor c	res 5,c	cpd
170	inverse E	AA	xor d	res 5,d	ind
171	inverse F	AB	xor e	res 5,e	outd
172	inverse G	AC	xor h	res 5,h	
173	inverse H	AD	xor l	res 5,l	
174	inverse I	AE	xor (hl)	res 5,(hl)	
175	inverse J	AF	xor a	res 5,a	
176	inverse K	B0	or b	res 6,b	ldir
177	inverse L	B1	or c	res 6,c	cpir
178	inverse M	B2	or d	res 6,d	inir
179	inverse N	B3	or e	res 6,e	otir
180	inverse O	B4	or h	res 6,h	
181	inverse P	B5	or l	res 6,l	
182	inverse Q	B6	or (hl)	res 6,(hl)	
183	inverse R	B7	or a	res 6,a	
184	inverse S	B8	cp b	res 7,b	lddr
185	inverse T	B9	cp c	res 7,c	cpdr
186	inverse U	BA	cp d	res 7,d	indr
187	inverse V	BB	cp e	res 7,e	otdr
188	inverse W	BC	cp h	res 7,h	
189	inverse X	BD	cp l	res 7,l	
190	inverse Y	BE	cp (hl)	res 7,(hl)	
191	inverse Z	BF	cp a	res 7,a	
192	CODE	C0	ret nz	set 0,b	
193	VAL	C1	pop bc	set 0,c	
194	LEN	C2	jp nz,NN	set 0,d	
195	SIN	C3	jp NN	set 0,e	
196	COS	C4	call nz,NN	set 0,h	
197	TAN	C5	push bc	set 0,l	
198	ASN	C6	add a,N	set 0,(hl)	
199	ACS	C7	rst 0	set 0,a	
200	ATN	C8	ret z	set 1,b	
201	LOG	C9	ret	set 1,c	
202	EXP	CA	jp z,NN	set 1,d	

Code	Character	Hex	Z80 assembler	– after CBh
203	INT	CB		set l,e
204	SQR	CC	call z,NN	set l,h
205	SGN	CD	call NN	set l,l
206	ABS	CE	adc a,N	set l,(hl)
207	PEEK	CF	rst 8	set 1,a
208	USR	D0	ret nc	set 2,b
209	STR$	D1	pop de	set 2,c
210	CHR$	D2	jp nc,NN	set 2,d
211	NOT	D3	out N,a	set 2,e
212	AT	D4	call nc,NN	set 2,h
213	TAB	D5	push de	set 2,l
214	**	D6	sub N	set 2,(hl)
215	OR	D7	rst 16	set 2,a
216	AND	D8	ret c	set 3,b
217	<=	D9	exx	set 3,c
218	>=	DA	jp c,NN	set 3,d
219	<>	DB	in a,N	set 3,e
220	TEMPO	DC	call c,NN	set 3,h
221	MUSIC	DD	prefixes instructions using ix	set 3,l
222	SOUND	DE	sbc a,N	set 3,(hl)
223	BEEP	DF	rst 24	set 3,a
224	NOBEEP	E0	ret po	set 4,b
225	LPRINT	E1	pop hl	set 4,c
226	LLIST	E2	jp po,NN	set 4,d
227	STOP	E3	ex (sp),hl	set 4,e
228	SLOW	E4	call po,NN	set 4,h
229	FAST	E5	push hl	set 4,l
230	NEW	E6	and N	set 4,(hl)
231	SCROLL	E7	rst 32	set 4,a
232	CONT	E8	ret pe	set 5,b
233	DIM	E9	jp (hl)	set 5,c
234	REM	EA	jp pe,NN	set 5,d
235	FOR	EB	ex de,hl	set 5,e
236	GOTO	EC	call pe,NN	set 5,h
237	GOSUB	ED		set 5,l
238	INPUT	EE	xor N	set 5,(hl)
239	LOAD	EF	rst 40	set 5,a
240	LIST	F0	ret p	set 6,b
241	LET	F1	pop af	set 6,c
242	PAUSE	F2	jp p,NN	set 6,d
243	NEXT	F3	di	set 6,e
244	POKE	F4	call p.NN	set 6,h
245	PRINT	F5	push af	set 6,l
246	PLOT	F6	or N	set 6,(hl)
247	RUN	F7	rst 48	set 6,a
248	SAVE	F8	ret m	set 7,b
249	RAND	F9	ld sp,hl	set 7,c
250	IF	FA	jp m,NN	set 7,d
251	CLS	FB	ei	set 7,e
252	UNPLOT	FC	call m,NN	set 7,h
253	CLEAR	FD	prefixes instructions using iy	set 7,l
254	RETURN	FE	cp N	set 7,(hl)
255	COPY	FF	rst 56	set 7,a